UNIVERSIDAD CENTROCIDENTAL
"LISANDRO ALVARADO"

SISTEMA PARA LA DETERMINACIÓN DE FLUJO EN UN CANAL VEHICULAR BASADO EN PROCESAMIENTO DE IMÁGENES Y REDES NEURONALES

ANTONIO MANUEL FERRAZ CARMONA

Barquisimeto, 2007

UNIVERSIDAD CENTROCIDENTAL “LISANDRO ALVARADO”
DECANATO DE CIENCIAS Y TECNOLOGÍA
POSTGRADO DE CIENCIAS DE LA COMPUTACIÓN
MENCIÓN: INTELIGENCIA ARTIFICIAL

SISTEMA PARA LA DETERMINACIÓN DE FLUJO EN UN CANAL VEHICULAR BASADO EN PROCESAMIENTO DE IMÁGENES Y REDES NEURONALES

Trabajo presentado para optar al grado de
Magíster Scientiarum

Por: ANTONIO MANUEL FERRAZ CARMONA

Barquisimeto, 2007

A mis Padres, Jaime y Petra Maria
por ser los seres especiales que Dios
me otorgó para guiar mi camino

AGRADECIMIENTOS

Agradezco principalmente a **Dios** por ser el mejor amigo, que con sus concejos, apoyo y sobre todo su misericordia, me formaron día a día, llenándome de obsequios tan grandes que no alcanzaran los libros para escribirlos; entre ellos están: la vida, la salud que poseo, mis padres, hermanos, tíos, familiares y amigos.

A mi **Madre Celestial María**, porque tomado de su mano me ha guiado desde pequeño por el camino de la vida, brindándome amor y protección en todo momento y especialmente en las dificultades.

A mis Padres: **Jaime y Petra**, por el amor que me han brindado desde que supieron de mi existencia, por el esfuerzo y sacrificio que han realizado para proporcionarme todo lo necesario en la vida.

A mis hermanos: **Jaime** y **Yaimar**, por estar siempre a mi lado brindándome su ayuda y concejos cuando más lo necesité, compartiendo conmigo momentos muy especiales que siempre llevaré en mi mente.

A **María Alejandra**, porque es un ser especial que tiene un lugar muy grande en mi corazón que con su amor, concejo, confianza y fortaleza en todo los momentos de mi vida me ha animado a seguir adelante.

Al **Ing. Fernando Del Bufalo**, por brindar su apoyo, conocimiento, tiempo y amistad incondicional que me motivan a seguir siempre adelante.

Al **Ing. Carlos Lameda y su esposa** que me brindaron conocimiento y su valioso tiempo para solucionar los problemas presentados.

A **todas aquellas personas**, que durante la realización de este trabajo, me facilitaron la información y los recursos que requería para lograr con éxito mi investigación.

A todos, GRACIAS.

ÍNDICE

ÍNDICE DE TABLAS

PÁG.

ÍNDICE DE FIGURAS

PÁG.

UNIVERSIDAD CENTROCCIDENTAL "LISANDRO ALVARADO"
DECANATO DE CIENCIAS
POSTGRADO DE CIENCIAS DE LA COMPUTACIÓN
MENCIÓN: INTELIGENCIA ARTIFICIAL

SISTEMA PARA LA DETERMINACIÓN DE FLUJO EN UN CANAL VEHICULAR BASADO EN PROCESAMIENTO DE IMÁGENES Y REDES NEURONALES

Autor: Antonio Manuel Ferraz Carmona
Tutor: Carlos Ignacio Lameda Montero

RESUMEN

En este trabajo se presenta el diseño de un sistema para determinar el flujo en un canal vehicular que emplea procesamiento de imágenes y redes neuronales. La estructura del sistema consta de tres bloques: detector de objetos en movimiento, extractor de características y clasificador/contador de vehículos.

Para el detector de objetos en movimiento se empleó la técnica de substracción de fondo, y un procedimiento en el cual el fondo evoluciona con el tiempo empleando una mascara de actualización y computación acumulativa. Con ello se mejoró el tiempo de respuesta, en comparación con el uso de substracción de fondo pero empleando una función para la adaptación del fondo.

Para el extractor de características se implementó un filtro Sobel y aprendizaje a través de una red de Kohonen. La propiedad de agrupamiento de la red de Kohonen permitió extraer los pixeles característicos de los objetos en movimiento sin importar las dimensiones del objeto, y establecer un número de pixeles fijo, que sirven como entrada a una red de retropropagación, la cual es entrenada para la identificación de los objetos en movimiento.

El sistema pudo contar vehículos aún cuando existiese un ruido por inclinación o desviación de la cámara. El empleo de un dispositivo de captura de video de baja resolución o alta resolución no afecta el resultado cuando hay buena iluminación.

La implementación de este tipo de sistema puede proporcionar al ingeniero de tránsito información necesaria para el cálculo y ajuste de las luces de semáforos en forma automática y tiempo real, y con ello ayudar al mejoramiento del flujo vehicular.

Palabras Clave: Redes neuronales, Computación acumulativa, flujo vehicular, procesamiento de imágenes.

INTRODUCCIÓN

Un problema al que se enfrentan los ingenieros de tránsito, es el cálculo de los intervalos de tiempo de encendido de las luces de un controlador de tránsito en una intersección. Por tal razón, se han diseñado controladores de tránsito que permiten ejecutar diferentes planes de temporización para cada día y a distintas horas, todo con el objeto de mejorar el flujo vehicular.

Hasta el momento, el método para el cálculo de los tiempos óptimos de encendido para cada luz de la intersección, se hace contando los vehículos en las diferentes horas del día, y luego se determina el flujo vehicular y se estima un valor adecuado para la luz.

Dicho conteo lo realizan operadores que realizan un registro de los vehículos por hora y el tipo de vehículos que transitan. El principal problema es el costo de tener operadores contabilizando los vehículos. Para reducir costos, se limita el estudio a un rango de tiempo, y como sabemos el flujo vehicular es variante con el tiempo. De esta forma, los cálculos de los tiempos son correctos durante un intervalo, pero luego de un tiempo (semanas o meses) el congestionamiento de la intersección comienza a observarse con frecuencia.

Es notable la necesidad de diseñar un sistema que permita determinar la cantidad y los tipos de vehículos que transitan, y de este modo ajustar los intervalos de duración de las luces para que cada vez que el flujo vehicular cambie se evite el congestionamiento.

Tradicionalmente se han usado los sensores magnéticos que permiten detectar la presencia de vehículos pero requieren ser instalados inmersos en la capa de asfalto.

Esto limita su aplicación, ya que existen lugares en los que no es posible perforar el asfalto (ejemplo una calle histórica), motivando la búsqueda de una tecnología alternativa.

En 1978 la NASA realizó un estudio en el que concluyó que la visión artificial ofrece una alternativa factible para la detección de vehículos. Luego con los avances en el procesamiento de imágenes y en la electrónica han incrementado la aplicación del video para la detección de objetos (sobre todo en los sistemas de vigilancia).

Durga (1999), Fernández (2001) y Cheung y Kamath (2003) han optado por utilizar técnicas que utilizan la información del movimiento para la detección de los objetos en una escena de video que permiten aumentar la rapidez de la detección, ya que se procesa sólo las regiones de la imagen donde ocurrió algún movimiento. Pero en aplicaciones en exteriores, la naturaleza se encarga de proporcionar falsos movimientos, como son la variación de la luz (sombra), movimiento de las ramas de los árboles, problemas de oclusión, apertura y otros causados por las limitaciones del sensor de captura de video, ya que se captura la imagen bidimensionalmente cuando en realidad es tridimensional.

En este trabajo se propone el diseño de un sistema para la determinación del flujo en un canal vehicular, constituido en tres bloques: detección de movimiento, extracción de características y detector o clasificador de vehículos.

Para el detector de objetos en movimiento se evaluó la técnica de computación acumulativa e interacción lateral propuesta por Fernández (2001), pero su tiempo de respuesta y la ausencia de información sobre la interacción lateral de la capa 2 y 3 del modelo, dio motivo para escoger la técnica de substracción de fondo con fondo evolutivo en el tiempo. Luego a los objetos en movimiento detectados se le aplicó el filtro Sobel, para extraer los contornos del objeto que son utilizados para realizar el aprendizaje de la red de Kohonen. Luego del aprendizaje, cada neurona representa

una característica extraída del objeto, conformándose de este modo el bloque extractor de características.

Por último las características extraídas son las entradas de una red de retropropagación que identifica si el objeto en movimiento es algún tipo de vehículo o no, para de este modo realizar el conteo del vehículos en el canal por unidad de tiempo obteniéndose el flujo vehicular.

CAPITULO I
EL PROBLEMA

Planteamiento del problema

El desarrollo tecnológico e industrial ha permitido un crecimiento urbano, a tal punto que los problemas en las intersecciones viales motivaron a los investigadores a diseñar y desarrollar controles de tránsito, que permiten alargar o recortar sus tiempos dependiendo del flujo vehicular. Por lo tanto, el control necesita un medio para adquirir la información necesaria (presencia de vehículo en el canal), para realizar los respectivos ajustes en la luz y de este modo evitar el congestionamiento en la intersección.

Tradicionalmente se han usado los sensores magnéticos, donde el lazo inductivo es el predominante (actualmente usado en algunas intersecciones de la ciudad de Barquisimeto). Los detectores de lazo inductivo permiten detectar la presencia del vehículo, pero se requiere instalarlo inmerso en la capa de asfalto y esto limita su función al no poder ser reubicado dependiendo de la necesidad. Otra desventaja de éste, es que no distingue entre un vehículo pesado o liviano, ya que cualquier metal lo activa y además los vehículos deben pasar sobre el sensor para ser detectados lo que ocasiona errores en la medición. Por tal razón, los investigadores han estado explorando continuamente una tecnología alternativa para el desarrollo de sensores magnéticos.

En 1978, la NASA (Jet Propulsion Laboratory) realizó un estudio en el que concluyeron que la visión artificial ofrece una alternativa factible para la detección de

vehículos (Carl, 1978). Esto fue verificado por investigadores en la Universidad de Minnesota al implementar un sistema para la detección de vehículos en secuencias de video para los usos prácticos en un control de tráfico en tiempo real (Michalopoulos, 1991). Con los avances en las técnicas de procesamiento de imágenes y la electrónica, se han buscado soluciones que han incrementado su aplicación y la aceptación por los ingenieros de tránsito; pero el costo y la confiabilidad de los sistemas para la determinación de flujo vehicular han limitado su aplicación.

Actualmente se emplean las cámaras instaladas en los circuitos cerrados de monitoreo del tráfico vehicular como el medio de captura del video, que luego es procesado por un módulo que permita indicar accidentes y las vías congestionadas. Un ejemplo es el proyecto que está desarrollando la Universidad Politécnica de Valencia de España a través de un convenio con el Ayuntamiento en la que se pretende convertir las 600 cámaras de la ciudad de Valencia (España) en cámaras inteligentes, donde ya tienen implementado 16 módulos Inteligentes(P.G., 2005).

Por ello, surge la siguiente pregunta, ¿será que una solución confiable y robusta para el sistema para la determinación de flujo vehicular puede ser encontrada a través de la aplicación de técnicas de procesamiento de imágenes y redes neuronales?

Objetivo General

Diseñar un sistema para la determinación de flujo en un canal vehicular basado en procesamiento de imágenes y redes neuronales.

Objetivos Específicos

- Determinar un método para detectar vehículos en secuencias de imágenes.
- Establecer una estructura general para un sistema de determinación de flujo vehicular que utilice redes neuronales.

- Determinar una técnica adecuada de detección y extracción de características de los objetos en movimiento.

- Diseñar un módulo para clasificar las características de los objetos en movimiento empleando redes neuronales.

- Probar la viabilidad del sistema mediante la implementación de un prototipo.

Justificación

El crecimiento del flujo vehicular en las intersecciones de una ciudad, ha ocasionado que los métodos de temporización actuales para el control de las luces de una intersección produzcan congestionamiento cuando ocurre cualquier imprevisto. Por ello es necesario dotar a los controles de tránsito de un sistema que le permita ajustar los tiempos para solventar el problema en tiempo real, así evitar el congestionamiento y el estrés de los ciudadanos**.**

Los sensores actuales presentan el problema que no distingue entre los tipos de vehículos, además de que cualquier metal lo acciona ocasionando errores en la medición. Estos los solventa la visión artificial ya que se pueden clasificar los

vehículos en el área a detectar y realizar un seguimiento de éstos para así obtener una medición precisa.

Como en la actualidad se está implementando circuitos cerrados en las principales vías e intersecciones de la ciudad (casco historico de barquisimeto), se facilita la aplicación del sistema al poder utilizar estos circuitos.

Este sistema puede realizar la función de todos los sensores tradicionales de una intersección, ya que ubicando la cámara de forma que capture toda la intersección y configurando las regiones por canal, se lograría captar y medir todos los canales a la vez permitiendo la disminución en los costos de la instalación de sensores.

Alcances y Limitaciones

Alcances

Se diseñó un sistema para la determinación del flujo vehicular basado en el procesamiento de imágenes y redes neuronales. El mismo tendrá una estructura de tres bloques: detector de movimiento, extractor de características y clasificador/ contador de vehículos (ver figura 1).

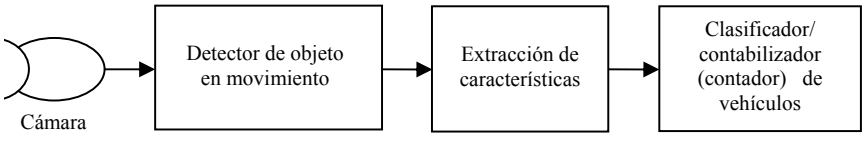

Figura 1. Diagrama general del sistema para la determinación de flujo vehicular.

El detector de objetos en movimiento procesa la imagen de la cámara y determina las regiones o siluetas de los objetos en movimiento. Este bloque tiene que enfrentarse a los problemas del ruido de la imagen, movimiento de las ramas de los árboles o sombra de una nube, condiciones de iluminación entre otras.

El extractor de características procesa las regiones o siluetas suministradas por el detector de objetos en movimiento para determinar las características del objeto que luego serán usadas para su clasificación.

El clasificador/contador de vehículos realiza una comparación de las características de los objetos con un conocimiento previo (base de datos) para clasificarlos y de este modo detectar los vehículos. Así mismo contabilizará los vehículos por unidad de tiempo.

Limitaciones

Entre las dificultades que se encontraron están:

Los sensores de video necesitan un requerimiento de luz para que la imagen capturada sea la adecuada para la detección de vehículos, por lo que el uso de una Cámara Web o la Cámara Sony no permitió detectar objetos en la noche y al atardecer los vehículos oscuros no fueron detectados. Esta limitación se puede solventar empleando cámaras de video con auto iris o visión infrarroja.

La ubicación del sensor de captura no fue la mas adecuada, ya que en algunos canales vehiculares ocurrían solapamientos de vehículos, por lo que una medición en

este canal muestra un error. Lo que es solucionable ubicando el sensor sobre el canal vehicular.

CAPITULO II
MARCO TEÓRICO

Antecedentes

Un estudio pionero en el uso de la visión artificial para el sensado de tráfico vehicular fue realizado por el Jet Propulsion Laboratory de la NASA (Carl, 1978) y el primer sistema práctico para la detección de vehículos mediante una secuencia de imágenes fue desarrollado por la Universidad de Minessota (Michalopoulos, 1991).

Durga (1999), diseñó una arquitectura para un detector vehicular que emula el detector de lazo inductivo. La detección del vehículo se realiza aplicando filtros digitales sobre un área de la imagen que es la zona del lazo, y se emplea hardware para regular la intensidad de iluminación para solventar el efecto de los cambios de iluminación en la escena. En la técnica se emplea la suposición de que los vehículos a detectar se encuentran en movimiento en el área correspondiente con el canal vehicular en estudio.

Klette y otros (2001) emplean la técnica de sustracción de fondo para la detección de vehículos en movimiento y para solucionar el problema de iluminación y del ruido de la señal de video utilizan un algoritmo que adapta la imagen de fondo en cada instante de tiempo. Por lo que un modelo adecuado para el sistema a desarrollar se tiene que adaptar a los cambios de iluminación y al ruido del video.

Fernández (2001) aplicó un modelo neuronal para la obtención de siluetas de objetos en movimientos en secuencias de imágenes. Se plantea un modelo multicapa que emplea la interacción lateral en computación acumulativa que solucionan los

problemas de apertura, oclusiones y movimientos debidos al fondo que son los principales problemas al detectar objetos en movimiento en una secuencia de imágenes.

Cheung y Kamath (2003) realizaron una comparación entre 5 algoritmos aplicados sobre una secuencia de tráfico urbano. De este trabajo se puede apreciar que la técnica más sencilla es la diferencia consecutiva de imágenes pero es la más afectada por los cambios de iluminación y el ruido en la señal de video**,** por lo que es necesario emplear una técnica que solucione los efectos de la iluminación y el ruido en la señal de video, ya que estos son los que acarrean más errores en los sistemas de detección de vehículos empleados.

Bogomolov y otros (2003) después de realizar la segmentación del movimiento en la secuencia de imágenes utilizando la técnica de sustracción de fondo, se extraen las características del objeto en movimiento que es comparada con una base de datos con el fin de clasificar el objeto.

Cooper y otros (2003) desarrollaron un sistema de reconocimiento y detección de vehículos inspirados en el movimiento rápido de los ojos y la selección selectiva. Este enfoque no toma la información del movimiento para realizar la detección de los objetos, para esto se emplea un algoritmo de búsqueda que simula la manera que el humano busca los objetos en una imagen (Saccadic). Para la identificación de los vehículos estos son modelados como un conjunto de características ordenadas de acuerdo a un punto de fijación.

De los trabajos descritos anteriormente, se puede establecer las siguientes ideas para desarrollar un sistema que permita detectar vehículos en movimiento:

- *Centrar la atención en las regiones donde ocurrió movimiento en el canal vehicular***.** Es importante emplear una técnica de segmentación adecuada. En

este caso se puede usar el modelo propuesto por Fernández (2001), ya que es una solución que emplea conceptos de inteligencia artificial y se adapta al objetivo de este trabajo. Aunque también se puede observar que el modelo de substracción de fondo es ampliamente usado para la detección de movimiento, debido a su simplicidad en el procesamiento haciéndolo adecuado para aplicaciones en tiempo real.

- *Extraer las características de los objetos en movimiento*. Se puede considerar el empleo de los bordes horizontales y verticales, porque son menos afectados por los cambios de iluminación (Coorper y otros (2003)).

- *Comparar las características extraídas de los objetos en movimiento con una base de conocimiento, y así determinar si el objeto puede ser clasificado como un vehículo*.

Bases Teóricas

DETECCIÓN DE OBJETOS EN MOVIMIENTO

La detección del movimiento en la secuencia de video permite enmarcar los objetos que se encuentran en la imagen, facilitando de este modo la extracción de las características del objeto y disminuyendo el procesamiento de los bloques siguientes del detector. Por lo tanto, el éxito del sistema radica en la selección de una buena técnica para tal fin.

Un buen detector de movimiento debe solventar los siguientes problemas:

- ***Apertura*.** Resolución unívoca del problema cuando no hay suficiente variación en el nivel de gris en la región estudiada de la imagen.

- ***Límites del movimiento*.** Superposición de objetos al ser proyectados sobre el espacio bidimensional, dado que algunos puntos del espacio tridimensional con diferente movimiento pueden proyectarse sobre un mismo punto en el plano de la imagen. Esto genera una discontinuidad en el campo del movimiento espacial, ya que esta región contiene una serie de movimientos distintos.

- ***Oclusión*.** Efecto en el que un fondo previamente cubierto se descubra o viceversa. En estas regiones no existe correspondencia entre los datos de diferentes imágenes y por ello, el campo movimiento no estará definido. Ocurren situaciones similares cuando hay un cambio de escena o cuando entran y salen nuevos objetos de la escena.

- ***Movimiento del fondo de la imagen.*** Falso movimiento del fondo de la imagen cundo un objeto en movimiento libera una zona de la imagen. En la figura 2 podemos apreciar el movimiento de un autobús a través de dos imágenes de una misma secuencia, así como la imagen diferencia de ambas. En la región con etiqueta (A) tenemos el movimiento detectado debido al objeto, mientras que (B) el falso movimiento debido al fondo de la imagen.

- **Velocidad del movimiento.** Se presenta cuando la intersección de un elemento en imágenes consecutivas no es vacía, lo que impide detectar movimiento en la zona de intersección de un objeto homogéneo, pero que sabemos que pertenecen al objeto (ver figura 3).

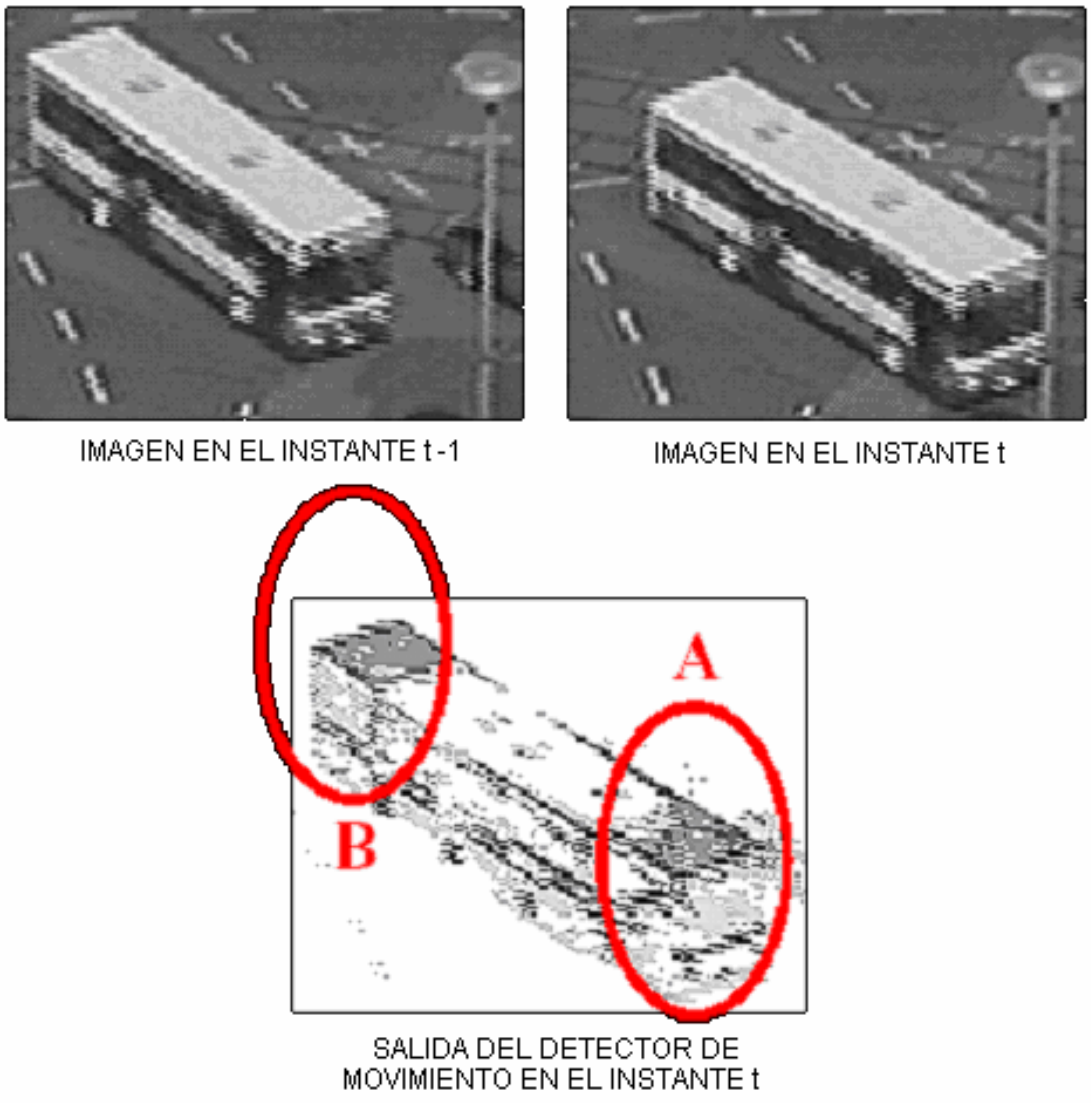

Figura 2. Imágenes consecutivas del movimiento de un autobús en la parte superior e imagen diferencia en la parte inferior, donde A es el movimiento detectado debido al objeto y B es el falso movimiento debido al fondo de la imagen.

Imágenes tomadas de la tesis doctoral de Fernández (2001). *Modelos De Interacción Lateral En Computación Acumulativa Para La Obtención De Siluetas*. Pág. 126.

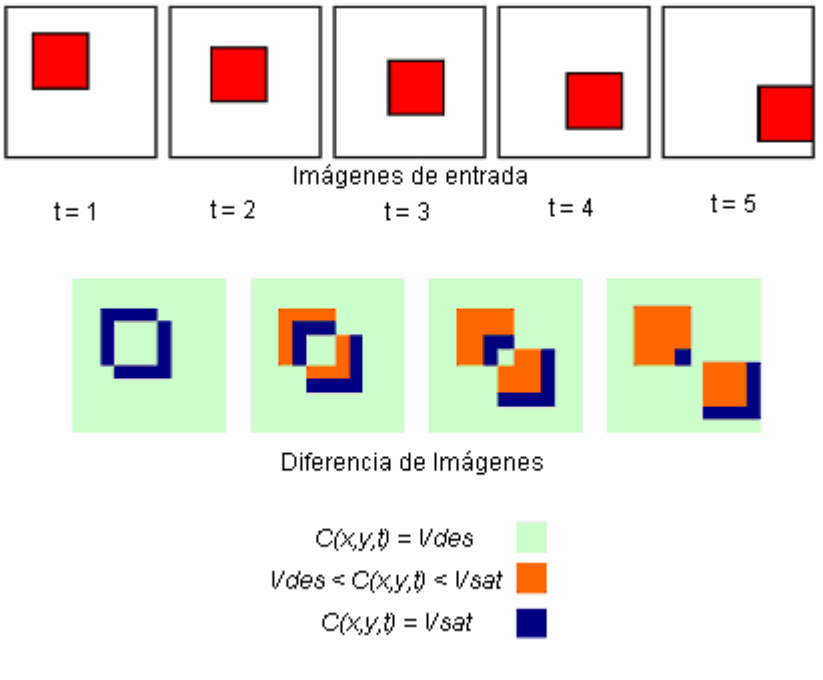

Figura 3. Obtención de los valores de carga.

Imagen tomada de la tesis doctoral de Fernández (2001). *Modelos De Interacción Lateral En Computación Acumulativa Para La Obtención De Siluetas*. pg. 18.

A continuación se describen brevemente los métodos de detección de movimiento mas usados.

Métodos basados en la correlación de características

Se basa en la extracción de un conjunto limitado de primitivas de las imágenes y su correlación en el tiempo utilizando algún tipo de estrategia de búsqueda. Estas primitivas (puntos, líneas y bordes, esquinas, entre otros) corresponden a características distintivas en la escena. Por ejemplo Cooper y otros (2003) utilizaron los bordes horizontales y verticales para la extracción de características y el algoritmo de búsqueda Saccadic.

Una vez encontrada las posiciones relativas de las características que se corresponden a lo largo de las tramas y a partir de un amplio conocimiento de la geometría proyectiva, puede ser estimada la posición relativa tridimensional y el movimiento de los objetos bajo ciertas condiciones. La complejidad del problema de la correspondencia entre características puede decrementarse utilizando técnicas de seguimiento bajo la suposición que el movimiento de cada una de las características es pequeño entre tramas consecutivas. Esto permite ofrecer predicciones acerca de las posiciones futuras de las características basadas en la historia de sus movimientos. Ya que la complejidad de la correlación crece exponencialmente con el número de características, estos métodos utilizan pocas características, aunque altamente discriminatorias.

Métodos basados en el gradiente

Los métodos basados en el gradiente estiman los gradientes espacial y temporal en cada punto de la imagen, usando directamente la ecuación del flujo óptico, e imponiendo restricciones normalmente de suavizado para identificar el movimiento de un modo unívoco.

El flujo óptico es la distribución de las velocidades aparentes del movimiento de formas brillantes en una imagen (Horn y Schunck, 1989). El flujo óptico, al provenir del movimiento relativo de los objetos y del observador, puede revelar una información importante de los objetos observados y el valor de cambios en la orientación. Las discontinuidades en el flujo óptico pueden igualmente ayudar a segmentar las imágenes en regiones que se corresponden con objetos diferentes.

Además del flujo óptico, el método se apoya en dos suposiciones: la iluminación global permanece constante en el tiempo y las superficies tienen una extensión espacial (Kollnig y Nagel, 1996). Por tanto, los pixeles en una vecindad de imagen pequeña son susceptibles de corresponderse con puntos de la misma superficie tridimensional, ya que el movimiento proyectado de los puntos de una superficie 3D variará gradualmente. Una consecuencia de esta suposición es imponer una restricción de suavizado en el campo del flujo óptico.

La restricción de iluminación global constante en el tiempo asume que cualquier cambio observado en la intensidad de la imagen a lo largo del tiempo, es causado por el movimiento relativo entre la cámara y los objetos en la escena (Heeger, 1987). Ya que el movimiento es una cantidad diferencial, una aproximación a su estimación es la de medir el grado de cambio espacio-temporal en la función de intensidad de la imagen, por lo que esta restricción puede formularse como una ecuación diferencial en términos de las derivadas parciales de la función de intensidad de la imagen, conocida como la ***ecuación del flujo óptico.***

La estimación basada en el gradiente es computacionalmente eficiente, pero debido al uso de ventanas de análisis pequeñas sufre necesariamente del problema de la apertura y de desviaciones naturales de la suposición de constancia de los datos, causadas por ejemplo, por la presencia de ruido debido a la observación o cambios en las condiciones globales de iluminación.

Métodos basados en regiones

Los métodos basados en regiones, relacionan regiones entre las diferentes tramas por medio de la búsqueda de las mismas. Cada trama se divide en regiones

y el movimiento de cada región se encuentra buscando la misma región en la siguiente trama (minimizando alguna medida de error, como puede ser la suma de las diferencias cuadradas entre los niveles de gris). Para ello se utilizan ventanas de análisis mayores para prevenir los problemas del ruido y los efectos de la apertura (Bowden y KaewTraKulPong, 2001).

El alto costo computacional del procedimiento de búsqueda y su falta de resolución son las desventajas de este tipo de aproximación (Jones y Viola, 2001). En el sensor de vehículos no se sabe las dimensiones de los vehículos y además la región de un vehículo en una imagen crece a medida que este se acerca a la cámara, por lo que esta técnica no es adecuada.

Sustracción del fondo y detección de cambios

La detección de cambios, también conocido como método de diferencia de imágenes consecutivas, consiste en calcular la diferencia de la imagen (***imagen diferencia***) entre dos imágenes consecutivas, lo que es conseguido restando las intensidades de las imágenes en cada pixel.

La imagen diferencia suele umbralizarse obteniéndose una imagen binaria, resaltando aquellos pixeles donde ha ocurrido un cambio significativo en su intensidad.

Los pixeles resaltados corresponden a partes de un objeto en movimiento siempre que la cámara esté en un lugar fijo, con apertura fija y condiciones de iluminación constantes. Si el objeto en movimiento tiene una textura uniforme, los pixeles resaltados se corresponderán con los bordes de cabeza y de cola del

objeto. Si el objeto tiene una textura heterogénea, algunos píxeles internos también serán resaltados.

Los pixeles resaltados pueden agruparse por medio de técnicas de clasificación para obtener un conjunto de regiones. El proceso de una escena con uno o más objetos en movimiento separados en la imagen, resultará en regiones que se corresponderán con cada uno de los objetos en movimiento.

Otra técnica poderosa, la sustracción del fondo, se basa en la disponibilidad de una imagen de referencia del fondo. Esta imagen puede obtenerse adquiriendo la imagen desde una cámara fija cuando no hay objetos en movimiento en la misma. Pero esto no es viable en aplicaciones en exteriores, como es el caso del sensor de vehículos. Por ello, se han propuesto modos alternativos para la generación de la imagen de referencia del fondo (Davis y otros (1999), Stauffer y Grinson (1999) y Klette y otros (2001)).

El problema no solo se basa en la determinación de la imagen de referencia de fondo, ya que el fondo es susceptible a los cambios ambientales en la iluminación por lo que ésta se actualiza periódicamente.

La sustracción de las imágenes (y la umbralización) se realiza del mismo modo que en la técnica de detección de cambios y los pixeles resaltados resultantes se corresponden con los objetos de interés. Suponiendo una cámara estacionaria con condiciones de iluminación fijas y un buen contraste, el método puede usarse para segmentar los objetos en movimiento en una escena. Los pixeles resaltados conectados entre sí generalmente se corresponden con objetos separados y las pequeñas regiones pueden ser ignoradas. Sin embargo, cuando varios objetos en movimiento se solapan en la imagen o están demasiado próximos los unos de los otros, solamente se obtiene una región fusionada.

Estas técnicas son sensibles a las sombras, a los cambios en la iluminación, a las vibraciones de la cámara, al poco contraste, al ruido de la imagen y a las oclusiones.

Modelo de interacción lateral en computación acumulativa

Fernández (2001) planteó un modelo general de interacción lateral en computación acumulativa, así como toda la arquitectura multicapa que puede emplearse en toda una serie de diferentes aplicaciones del mundo real, ya que solventa los principales problemas al segmentar el movimiento en ambientes externos.

El modelo está compuesto en cinco capas, que se describen a continuación:

Capa 0: *Segmentación por bandas de nivel de gris*. Esta capa cubre la necesidad anteriormente descrita de segmentar la imagen en un conjunto preestablecido de *n* bandas de nivel de gris. En cada uno de sus elementos, su entrada será el valor de nivel de gris del pixel correspondiente de la imagen a cada cuadro de tiempo global *t*. Estos valores indican la pertenencia o no del pixel a cada una de las bandas de nivel de gris.

Capa 1: *Interacción lateral para la computación acumulativa*. Esta capa se encarga de la obtención del valor de permanencia (*PMk (x,y,t))* en base a una descomposición en bandas de nivel de gris. Tendremos *n* subcapas y cada una de ellas albergará por cada elemento el valor de la computación acumulativa presente en la escala de tiempos global *t*. La interacción lateral en esta capa irá orientada a reactivar la carga de permanencia de aquellos elementos con cierto valor de carga,

que estén directa o indirectamente conectados a elementos saturados. La carga de permanencia se ofrecerá a la siguiente capa como salida.

Capa 2: *Interacción lateral para la obtención de elementos de siluetas*. También está formada por *n* subcapas, y en ellas se realiza mediante interacción lateral un reparto de la carga entre todos los vecinos con una carga mínima y conectados e interconectados entre sí. Además de repartir la carga (*Ck (x,y,t))* en bandas de nivel de gris se consigue, a este nivel, diluir la carga debida al movimiento del fondo. La nueva carga obtenida en esta capa es ofrecida como salida hacia la capa 3.

Capa 3: *Interacción lateral para la fusión de objetos en movimiento*. Cada uno de los elementos de la capa tienen una entrada desde cada elemento correspondiente de las *n* subcapas de la capa 2. Esta capa tiene como finalidad la unificación de las siluetas. Para ello, toma las cargas de entrada de cada una de las bandas de nivel de gris y efectúa una fusión de dichos valores, obteniendo cada una de las siluetas de la imagen origen. Su salida es el conjunto de siluetas *S(x,y,t)*.

Capa 4: *Aplicación***.** La capa 4 es una capa multifuncional capaz de implementar toda una serie de aplicaciones específicas de alto nivel. Esta capa estaría compuesta por los módulos: extractor de características y el detector de vehículos, retomando la estructura general de un sensor de vehículos expuesta al principio.

EXTRACCIÓN DE CARACTERÍSTICAS DE UNA IMAGEN

El objetivo de la extracción de características de una imagen es resaltar ciertos aspectos o propiedades sobre la imagen, que faciliten el proceso de clasificación de los objetos.

Los contornos caracterizan las fronteras de los objetos, y por tanto son de gran utilidad para la segmentación e identificación de objetos en escenas (González y Woods, 1993).

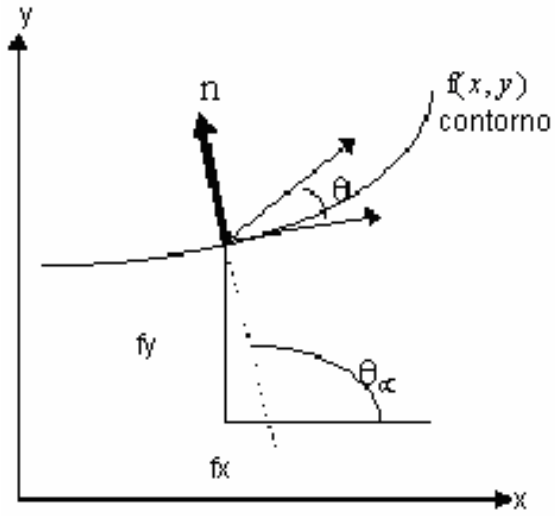

Figura 4. Gradiente de f(x,y)

Los puntos de contorno son zonas de pixeles en las que existe un cambio brusco de nivel de gris. Si pensamos en una imagen como una función continua *f(x,y)*, vemos que su derivada tiene un máximo local en la dirección del contorno. Es por esto, que las técnicas más usadas en la detección de contornos se basan en la medida del gradiente de *f* a lo largo de *r* en una dirección θ (Figura 4)

De lo anterior se puede inferir que el principal problema de la extracción de fronteras es el ruido; por lo que generalmente se utilizan filtros para eliminar el ruido antes de extraer los bordes de la imagen (Hue y otros, 2002).

En el tratamiento de imágenes, se han ideado operadores detectores de contornos basados en máscaras que representan aproximaciones en diferencias finitas

de los gradientes ortogonales f_x y f_y, que al realizar la convolución de estas con la imagen nos permite aproximar el gradiente de la imagen (Canny, 1986).

A continuación se describen los operadores (*máscaras*) para la detección de bordes más comunes:

- *Gradiente digital*

La forma habitual de establecer el gradiente de una imagen *U(m,n)* en un punto dado, es mediante el producto de la imagen por dos máscaras H1, H2 que representan la magnitud del gradiente en dos direcciones perpendiculares.

Tabla 1.

Operadores gradientes más comunes

Operadores gradiente más comunes	Dirección horizontal	Dirección vertical
Roberts	$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$
Smoothed (o Prewitt)	$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$
Sobel	$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$
Isotrópico	$\begin{bmatrix} -1 & 0 & 1 \\ -\sqrt{2} & 0 & \sqrt{2} \\ -1 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & -\sqrt{2} & -1 \\ 0 & 0 & 0 \\ 1 & \sqrt{2} & 1 \end{bmatrix}$

Por razones computacionales, la magnitud del gradiente se calcula como:

$$g(m,n) = |g1(m,n)| + |g2(m,n)| \qquad 1$$

Un pixel es declarado como perteneciente a un borde cuando *g(m,n)* excede un determinado valor umbral. Habitualmente se suele escoger este valor en función del histograma acumulado de *g(m,n),* de forma que sólo del 5 al 10% de los puntos de máximo gradiente sea declarado como borde (González y Woods, 1993).

En la tabla 1 se muestran las máscaras de uso más frecuente.

- *Compás (Compass Operators):*

Miden el gradiente en un número determinado de direcciones. Viene a ser una variante del operador del gradiente, sólo que con diferentes matrices (hasta ocho) para cada dirección. En la tabla 2 se muestran los operadores de Kirsch.

Tabla 2.

Operadores de Kirsch

K_0

-3	-3	5
-3	0	5
-3	-3	5

K_1

-3	5	5
-3	0	5
-3	-3	-3

K_2

5	5	5
-3	0	-3
-3	-3	-3

K_3

5	5	-3
5	0	-3
-3	-3	-3

K_4

5	-3	-3
5	0	-3
5	-3	-3

K_5

-3	-3	-3
5	0	-3
5	5	-3

K_6

-3	-3	-3
-3	0	-3
5	5	5

K_7

-3	-3	-3
-3	0	5
-3	5	5

Por ejemplo, un valor grande después de aplicar la primera máscara sobre un píxel implica que existe un borde en sentido vertical (por tanto, un gradiente horizontal) en el píxel donde se ha aplicado la máscara. Para encontrar los bordes, se aplica cada una de las máscaras en cada uno de los píxeles (se realiza una *convolución* con cada máscara). La respuesta del detector de bordes es el máximo de las respuestas de cada una de las ocho máscaras y la dirección sería p i/4 si K_i ha sido la máscara responsable de dicho máximo.

- *Laplaciano y cruces por cero*:

Los operadores anteriores funcionan bien con transiciones bruscas de nivel de gris, pero cuando la región de transición es muy ancha, resulta más efectivo aplicar las derivadas de segundo orden.

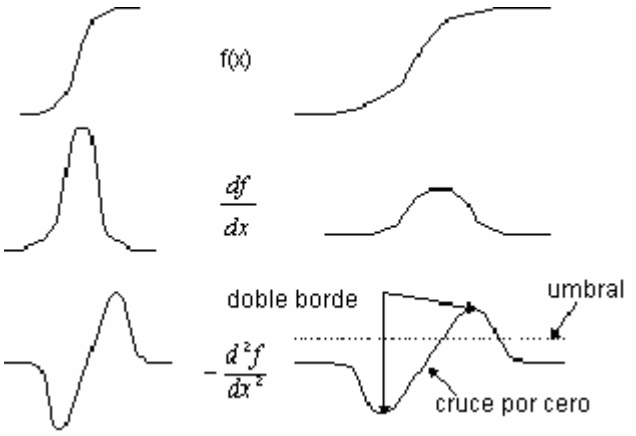

Figura 5. Primera y segunda derivada para detección de contornos por cruces por cero.

El operador Laplaciano está definido por la siguiente ecuación:

$$\nabla^2 f = \frac{df}{dx^2} + \frac{df}{dy^2} \qquad 2$$

Como se ve en la figura 5, al aplicar el umbral a la magnitud del gradiente, aparece un doble borde. Para resolver esto, lo que se hace es detectar los cruces por cero como bordes. En la práctica, para trabajar con imágenes discretas, se opera con las mascaras mostradas a continuación (González y Woods, 1993).

Tabla 3.

Operador Laplaciano

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \qquad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \qquad \begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

- *Gradiente estocástico*

El uso de máscaras estáticas ofrece una pobre respuesta en imágenes con ruido; para ello se idea este método que construye una máscara dinámica por cada pixel de la imagen, intentando reducir en la medida de lo posible el efecto que el ruido pueda provocar en el cálculo del gradiente. Obviamente es un algoritmo muy costoso en tiempo de cálculo, por lo que es desechado para aplicaciones en tiempo real.

REDES NEURONALES

Las Redes Neuronales Artificiales (RNA) o sistemas conexionistas son sistemas de procesamiento de la información cuya estructura y funcionamiento están inspirados en las redes neuronales biológicas (Jiménez y otros, 2001). Consisten en un conjunto de elementos simples de procesamiento llamados nodos o neuronas conectadas entre sí por conexiones que tienen un valor numérico modificable llamado peso.

En los últimos años, las redes neuronales artificiales han emergido como una potente herramienta para el modelado estadístico orientada principalmente al reconocimiento de patrones –tanto en la vertiente de clasificación como de predicción.

Entre los paradigmas de redes neuronales podemos destacar la red de retropropagación y los mapas autoorganizados de Kohonen, los cuales se explican a continuación.

Retropropagación

La red de retropropagación utiliza un esquema de aprendizaje supervisado, y su nombre está relacionado con la técnica para el ajuste de las neuronas en la etapa del aprendizaje. Esta red se ha utilizado satisfactoriamente en la clasificación de patrones y la estimación de funciones.

Cuando se presenta un patrón p de entrada X_p: $x_{p1},...,x_{pi},...,x_{pN}$, éste se transmite a través de los pesos w_{ji} desde la capa de entrada hacia la capa oculta (ver la figura 6). Las neuronas de esta capa intermedia transforman las señales recibidas mediante la aplicación de una función de activación proporcionando un valor que es transmitido a través de los pesos v_{kj} hacia la capa de salida, donde aplicando la misma operación que en el caso anterior, las neuronas de esta última capa proporcionan la salida de la red.

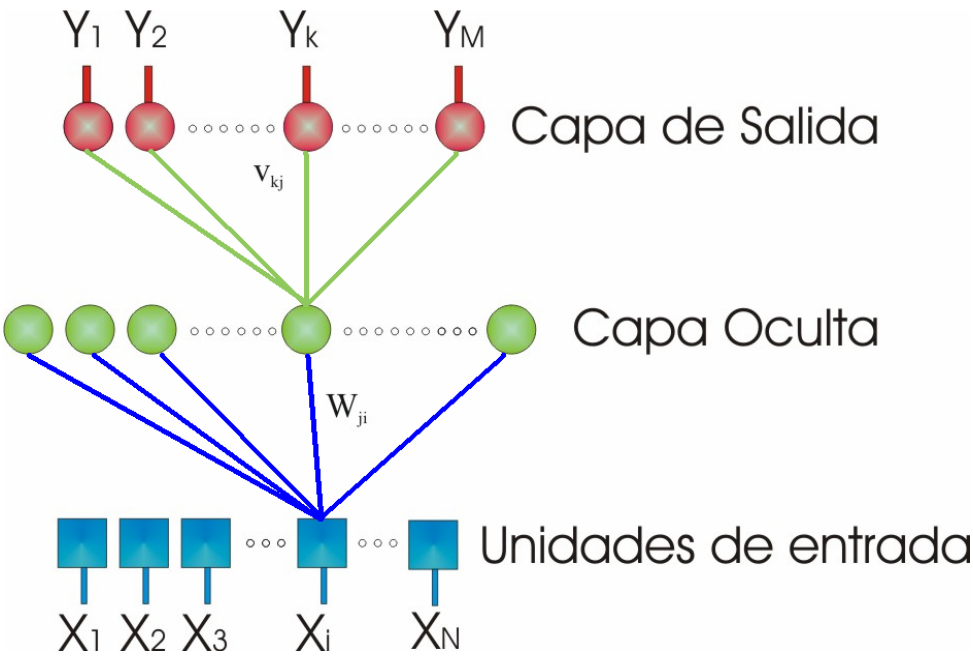

Figura 6. Red de retropropagación

Matemáticamente la entrada total o neta que recibe una neurona oculta j, net_{pj}, es:

$$net_{pj} = \sum_{i=1}^{N} w_{ji} x_{pi} + \theta_j^0 \qquad 3$$

donde Θ es el umbral de la neurona que se considera como un peso asociado a una neurona ficticia con valor de salida igual a 1.

El valor de salida de la neurona oculta *j,* b_{pj}, se obtiene aplicando una función $f(net_{pj})$ sobre su entrada neta:

$$b_{pj} = f(net_{pj}) \qquad 4$$

De igual forma, la entrada neta que recibe una neurona de salida *k,* net_{pk}, es:

$$net_{pk} = \sum_{j=1}^{L} v_{kj} b_{pj} + \theta_k \qquad 5$$

Por último, el valor de salida de la neurona de salida *k* (y_{pk}) es:

$$y_{pk} = f(net_{pk}) \qquad 6$$

La red neuronal para que cumpla el propósito para el que fue diseñado, es necesario entrenar la red. El objetivo que se persigue del algoritmo de aprendizaje es hacer mínima la discrepancia o error entre la salida obtenida por la red y la salida deseada por el usuario ante la presentación de un conjunto de patrones denominado grupo de entrenamiento. Por este motivo, se dice que el aprendizaje en las redes de retropropagación es de tipo supervisado.

La función de error que se pretende minimizar para cada patrón *p* viene dada por:

$$E_p = \frac{1}{2}\sum_{k=1}^{M}(d_{pk} - y_{pk})^2 \qquad 7$$

donde d_{pk} es la salida deseada para la neurona de salida k ante la presentación del patrón p. A partir de esta expresión se puede obtener una medida general de error mediante:

$$E = \sum_{p=1}^{P} E_p \qquad 8$$

La base matemática del algoritmo de retropropagación para la modificación de los pesos es la técnica conocida como gradiente decreciente (Hinton y otros, 1986). Teniendo en cuenta que E_p es función de todos los pesos de la red, el gradiente de E_p es un vector igual a la derivada parcial de E_p respecto a cada uno de los pesos. El gradiente toma la dirección que determina el incremento más rápido en el error, mientras que la dirección opuesta, determina el decremento más rápido en el error. Por tanto, el error puede reducirse ajustando cada peso en la dirección:

$$-\sum_{p=1}^{P} \frac{\partial E_p}{\partial w_{ji}} \qquad 9$$

Existe la posibilidad de que el aprendizaje converja en un mínimo local y no el mínimo global de la superficie del error. Por lo que en la práctica se aplica el algoritmo de aprendizaje una serie de veces inicializando los pesos en forma aleatoria cada vez, escogiéndose la mejor solución que generalmente coincide con el mínimo global.

A nivel práctico, la forma de modificar los pesos de forma iterativa consiste en aplicar la regla de la cadena a la expresión del gradiente y añadir una tasa de aprendizaje (η). Así, la variación del peso de una neurona de salida:

$$\Delta v_{kj}(n+1) = \eta \sum_{p=1}^{P} \delta_{pk} b_{pj} \qquad 10$$

donde

$$\delta_{pk} = (d_{pk} - y_{pk}) f'(net_{pk}) \qquad 11$$

y n indica la iteración. Y la variación del peso de una neurona oculta:

$$\Delta w_{ji}(n+1) = \eta \sum_{p=1}^{P} \delta_{pj} x_{pi} \qquad 12$$

donde

$$\delta_{pj} = f'(net_{pj}) \sum_{k=1}^{M} \delta_{pk} v_{kj} \qquad 13$$

Se puede observar que el error asociado a una neurona oculta *j,* viene determinado por la suma de los errores que se cometen en las *k* neuronas de salida que reciben como entrada la salida de esa neurona oculta *j*. De ahí que el algoritmo también se denomine *propagación del error hacia atrás*.

Existen dos modos de actualizar los pesos de la red. Por lotes donde la actualización se realiza después de haber presentado todos los patrones de entrenamiento, y por serie (on line) donde se actualizan los pesos tras la presentación de cada patrón de entrenamiento. Para este último es necesario presentar los patrones de entrenamiento de forma aleatoria.

Con el fin de acelerar el proceso de convergencia de los pesos, Hinton y otros (1986) sugirieron añadir en la expresión del incremento de los pesos un factor momento ($\propto$), el cual tiene en cuenta la dirección del incremento tomada en la iteración anterior. Así las variaciones de los pesos se calculan con las siguientes ecuaciones; en los pesos de las neuronas de salida

$$\Delta v_{kj}(n+1) = \eta\left(\sum_{p=1}^{P} \delta_{pk} b_{pj}\right) + \alpha \Delta v_{kj}(n) \qquad 14$$

y en los pesos de una neurona oculta

$$\Delta w_{ji}(n+1) = \eta\left(\sum_{p=1}^{P} \delta_{pj} x_{pi}\right) + \alpha \Delta w_{ji}(n) \qquad 15$$

Los mapas autoorganizados de Kohonen

En 1982 Teuvo Kohonen presentó un modelo de red denominado mapas autoorganizados o SOM (*Self-Organizing Maps*), basado en ciertas evidencias descubiertas a nivel cerebral y con un gran potencial de aplicabilidad práctica. Este tipo de red se caracteriza por poseer un aprendizaje no supervisado competitivo.

En el aprendizaje no supervisado (o autoorganizado) no existe ningún maestro externo que indique si la red neuronal está operando correcta o incorrectamente, pues no se dispone de ninguna salida objetivo hacia la cual la red neuronal deba tender. Así, durante el proceso de aprendizaje la red autoorganizada debe descubrir por sí misma rasgos comunes, regularidades, correlaciones o categorías en los datos de entrada, e incorporarlos a su estructura interna de conexiones.

Dentro del aprendizaje no supervisado existe un grupo de modelos de red caracterizados por poseer un aprendizaje competitivo. En el aprendizaje competitivo las neuronas compiten unas con otras con el fin de llevar a cabo una tarea dada. Con este tipo de aprendizaje, se pretende que cuando se presente a la red un patrón de entrada, sólo una de las neuronas de salida (o un grupo de vecinas) se active. Por tanto, las neuronas compiten por activarse, quedando finalmente una como neurona vencedora y anuladas el resto, que son forzadas a sus valores de respuesta mínimos.

El objetivo de este aprendizaje es categorizar ("*clusterizar*") los datos que se introducen en la red (Jiménez y otros, 2002). De esta forma, las informaciones similares son clasificadas formando parte de la misma categoría y, por tanto, deben activar la misma neurona de salida. Las clases o categorías deben ser creadas por la propia red, puesto que se trata de un aprendizaje no supervisado, a través de las correlaciones entre los datos de entrada.

Un modelo SOM está compuesto por dos capas de neuronas. La capa de entrada (formada por N neuronas, una por cada variable de entrada) se encarga de recibir y transmitir a la capa de salida la información procedente del exterior. La capa de salida (formada por M neuronas) es la encargada de procesar la información y formar el mapa de rasgos. Normalmente, las neuronas de la capa de salida se organizan en forma de mapa bidimensional como se muestra en la figura 7, aunque a veces también se utilizan capas de una sola dimensión (cadena lineal de neuronas) o de tres dimensiones (paralelepípedo).

Las conexiones entre las dos capas que forman la red son siempre hacia adelante; es decir, la información se propaga desde la capa de entrada hacia la capa de salida. Cada neurona de entrada i está conectada con cada una de las neuronas de salida j mediante un peso w_{ji}. De esta forma, las neuronas de salida tienen asociado un vector de pesos W_j llamado vector de referencia, debido a que constituye el vector prototipo (o promedio) de la categoría representada por la neurona de salida j.

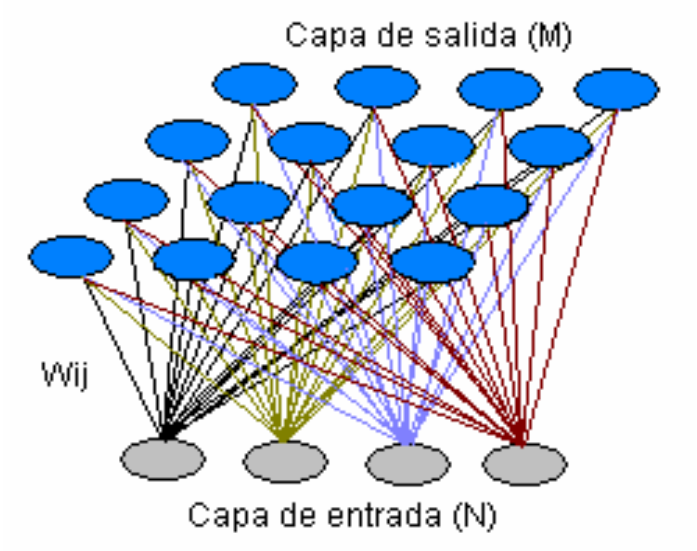

Figura 7. Arquitectura del SOM representado en un plano bidimensional.

Entre las neuronas de la capa de salida, puede decirse que existen conexiones laterales de excitación e inhibición implícitas, pues aunque no estén conectadas, cada una de estas neuronas va a tener cierta influencia sobre sus vecinas. Esto se consigue a través de un proceso de competición entre las neuronas y de la aplicación de una función vecindad.

En funcionamiento normal se presenta un patrón p de entrada X_p: $x_{p1},...,$ $x_{pi},...,x_{pN}$ al SOM entrenado, el cual es transmitido directamente desde la capa de entrada hacia la capa de salida. En esta capa, cada neurona calcula la similitud entre el vector de entrada X_p y su propio vector de pesos W_j o vector de referencia según una cierta medida de distancia o criterio de similitud establecido. Finalmente se declara vencedora la neurona cuyo vector de pesos es el más similar al de entrada.

Lo descrito anteriormente queda representado por la siguiente expresión matemática:

$$y_{pj} = \begin{cases} 1 \ si \ min \left| Xp + Wj \right| \\ 0 \text{ para el resto} \end{cases} \qquad 16$$

Donde y_{pj} representa la salida o el grado de activación de las neuronas de salida en función del resultado de la competición (1 = neurona vencedora, 0 = neurona no vencedora), $||Xp-W_j||$ representa una medida de similitud entre el vector o patrón de entrada $Xp: x_{p1},..., x_{pi},..., x_{pN}$ y el vector de pesos $Wj: w_{j1},..., w_{ji},...w_{jN}$, de las conexiones entre cada una de las neuronas de entrada y la neurona de salida *j*.

No existe un algoritmo de aprendizaje totalmente estándar para la red SOM. Sin embargo, se trata de un procedimiento bastante robusto ya que el resultado final es en gran medida independiente de los detalles de su realización concreta.

El algoritmo de aprendizaje propuesto por Kohonen trata de establecer, mediante la presentación de un conjunto de patrones de entrenamiento, las diferentes categorías que servirán durante la etapa de funcionamiento para realizar clasificaciones de nuevos patrones de entrada.

De forma simplificada, el proceso de aprendizaje se desarrolla de la siguiente manera.

Una vez presentado y procesado un vector de entrada, se establece a partir de una medida de similitud la neurona vencedora; esto es, la neurona de salida cuyo vector de pesos es el más parecido respecto al vector de entrada. A continuación, el vector de pesos asociado a la neurona vencedora se modifica de manera que se parezca un poco más al vector de entrada. De este modo, ante el mismo patrón de entrada, dicha neurona responderá en el futuro todavía con más intensidad. El proceso se repite para un conjunto de patrones de entrada los cuales son presentados repetidamente a la red, de forma que al final los diferentes vectores de pesos sintonizan con uno o varios patrones de entrada y, por tanto, con dominios

específicos del espacio de entrada. Si dicho espacio está dividido en grupos, cada neurona se especializará en uno de ellos, y la operación esencial de la red se podrá interpretar como un análisis de clusters.

Al finalizar el aprendizaje, el vector de referencia de cada neurona de salida se corresponderá con el vector de entrada que consigue activar la neurona correspondiente. En el caso de existir más patrones de entrenamiento que neuronas de salida, como en el ejemplo expuesto, más de un patrón deberá asociarse con la misma neurona, es decir, pertenecerán a la misma clase. En tal caso, los pesos que componen el vector de referencia se obtienen como un promedio (*centroide*) de dichos patrones.

Además de este esquema de aprendizaje competitivo, el modelo SOM incorpora relaciones entre las neuronas próximas en el mapa. Para ello, introduce una función denominada *zona de vecindad* que define un entorno alrededor de la neurona ganadora actual; su efecto es que durante el aprendizaje se actualizan tanto los pesos de la vencedora como los de las neuronas pertenecientes a su vecindad. De esta manera, el modelo SOM logra que neuronas próximas se sintonicen con patrones similares, quedando de esta manera reflejada sobre el mapa una cierta imagen del orden topológico presente en el espacio de entrada.

Un criterio de similitud muy utilizado es el de la *distancia euclídea* que viene dado por la siguiente expresión:

$$\min \left\| X_p - W_j \right\| = \min \sum_{i=1}^{N} (x_{pi} - w_{ji})^2 \qquad 17$$

De acuerdo con este criterio, dos vectores serán más similares cuanto menor sea su distancia.

Una medida de similitud alternativa más simple que la euclídea, es la correlación o producto escalar:

$$\min \left\| X_p - W_j \right\| = \max \sum_{i=1}^{N} x_{pi} \cdot w_{ji} \qquad 18$$

según la cual, dos vectores serán más similares cuanto mayor sea su correlación.

Identificada la neurona vencedora mediante el criterio de similitud, podemos pasar a modificar su vector de pesos asociado y el de sus neuronas vecinas, según la regla de aprendizaje:

$$\Delta w_{ji}(n+1) = \alpha(n)\left(x_{pi} - w_{j*i}(n)\right) \quad \text{para } j \in \text{Zona}_{j*}(n) \qquad 19$$

donde n hace referencia al número de ciclos o iteraciones, $\propto(n)$ es la tasa de aprendizaje, $Zona_{j*}(n)$ es la zona de vecindad alrededor de la neurona vencedora $_{j*}$ en la que se encuentran las neuronas cuyos pesos son actualizados. La tasa de aprendizaje se inicializa con un valor entre 0 y 1.

Tradicionalmente el ajuste de los pesos se realiza después de presentar cada vez un patrón de entrenamiento, como se muestra en la regla de aprendizaje expuesta. Sin embargo, se puede acumular los incrementos calculados para cada patrón de entrenamiento y actualizar los pesos a partir del promedio de incrementos acumulados, una vez presentados todos los patrones. Para evitar que la dirección del vector de pesos vaya oscilando de un patrón a otro y acelerar la convergencia de los pesos de la red.

En el proceso general de aprendizaje suelen considerarse dos fases. En la primera fase, se pretende organizar los vectores de pesos en el mapa. Para ello, se

comienza con una tasa de aprendizaje y un tamaño de vecindad grandes, para luego ir reduciendo su valor a medida que avanza el aprendizaje. En la segunda fase, se persigue el ajuste fino del mapa, de modo que los vectores de pesos se ajusten más a los vectores de entrenamiento. El proceso es similar al anterior aunque suele ser más largo, tomando la tasa de aprendizaje constante e igual a un pequeño valor (por ejemplo, 0.01) y un radio de vecindad constante e igual a 1.

CAPITULO III
MARCO METODOLÓGICO

Naturaleza de la Investigación

El presente trabajo está enmarcado dentro de la modalidad de proyecto factible; el cual está orientado a resolver problemas de ingeniería de tránsito.

La investigación se basa en el diseño de un sistema para la determinación de flujo en un canal vehicular, utilizando procesamiento de imágenes y redes neuronales.

El trabajo se efectuará en cuatro fases:

- Fase I Diagnóstico.
- Fase II Factibilidad.
- Fase III Diseño del sistema.
- Fase IV Implementación de un prototipo del sistema.

Fase I Diagnóstico

En esta fase se llevó a cabo la recolección de información, el análisis de los objetivos y características del problema planteado, generándose el siguiente procedimiento:

1. **Se diseño una estructura general del sistema para la determinación de flujo vehicular**.

Se realizó el análisis de la documentación y los objetivos planteados, generándose una estructura general de tres bloques para la determinación del flujo vehicular mostrado en la figura 1.

El diseño de cada bloque de la estructura busca disminuir la cantidad de información a ser procesada por el siguiente bloque, de forma que al clasificador sólo le llegan los *pixeles* característicos, disminuyéndose los requerimientos de procesamiento.

Como el tiempo de procesamiento es importante para el sistema, se evaluó el tiempo de respuesta de la implementación del prototipo con Visual Basic o C++ Builder. Para esto se utilizó el detector de movimiento en ambos lenguajes de programación, comprobándose que la implementación en C++ Builder fue aproximadamente 5 veces más rápida que la implementada con Visual Basic, razón que permitió elegir a C++ Builder como lenguaje para la realización del prototipo.

2. **Se capturaron los videos del canal vehicular**.

La ubicación de la cámara es importante, ya que se tiene que evitar que un vehículo oculte a otro ocasionando que el detector de movimiento lo detecte como un solo objeto (vehículos solapados), generándose un error en la medición. Por lo que la ubicación ideal es perpendicular o paralela al canal en una estructura alta como un poste o un edificio, esto permite capturar el canal desde una inclinación adecuada y evita solapar los vehículos en la imagen. Las imágenes utilizadas para la realización del prototipo, fueron tomadas a la altura del onceavo piso del edificio Metropolitan localizado en la Av. Venezuela entre calles 40 y 41 de la

ciudad de Barquisimeto. Se capturaron videos a diferentes horas del día, para evaluar al sistema con diferentes condiciones de iluminación (mañana, tarde y anochecer).

3. **Se generó los patrones de entrenamiento del sistema y el ajuste del sistema**.

El ajuste del sistema consiste en adaptar los umbrales hasta obtener la respuesta más adecuada del sistema. Al sistema se le presentó un video de entrada, luego se cambió un umbral y se volvió a presentar el video para comprobar si el cambio mejoraba la respuesta del sistema, repitiendo este procedimiento con cada umbral hasta encontrar una respuesta satisfactoria. Para lograr este ajuste se realizó un software (Figura 8) que permite hacer los ajustes de los umbrales, que a diferencia del prototipo final, muestra la imagen del video de entrada, la imagen del fondo generada, la imagen resultado de aplicar la sustracción del fondo, las regiones con objetos en movimiento detectado y los pixeles característicos que cada objeto extraído.

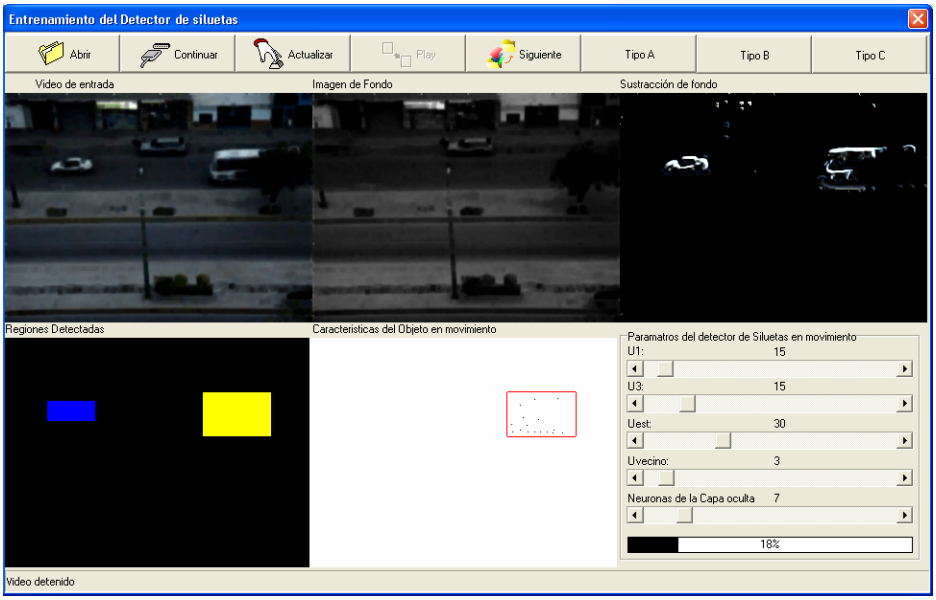

Figura 8. Ventana de ajuste de umbrales del sistema.

Una vez obtenidos los videos y ajustado el sistema, se procedió a extraer los objetos de tres videos (uno para cada condición de iluminación), se indicó si el objeto en movimiento es un vehículo y a que tipo de vehículo pertenece. Para esto se empleó el software mencionado en el párrafo anterior, haciéndose clic en el botón de acceso rápido "Siguiente" se extrajo las características del otro objeto en movimiento, y se almacenó en el archivo de patrones según el tipo de vehículo seleccionando el botón de acceso rápido "Tipo A", "Tipo B" o "Tipo C" (estos tipo de vehiculos los asigna el usuario según su necesidad, ejemplo, A no es un vehiculo, "B" es un vehiculo liviano y "C" es un vehiculo Pesado).

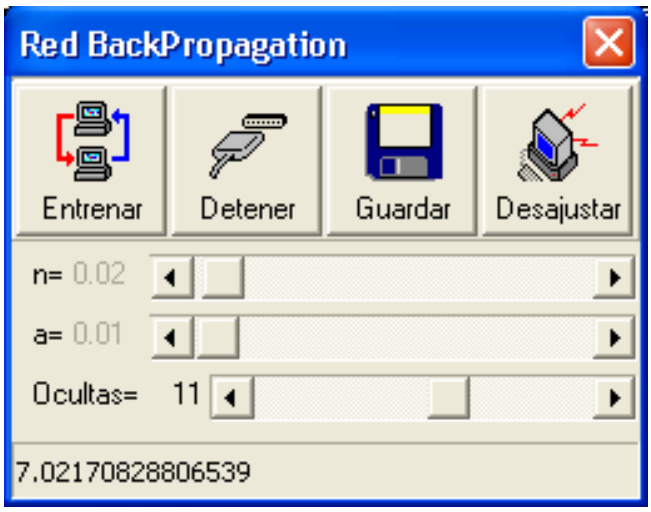

Figura 9. Ventana de entrenamiento de la red de retropropagación.

Luego se procedió a entrenar la red de retropropagación ajustando las constantes de aprendizaje y número de neuronas ocultas, hasta encontrar un error aceptable de la red (clasificador). Se diseñó un software (Figura 9) que entrena la red de retropropagación y genera un archivo que es utilizado por el prototipo con los pesos de las neuronas.

4. **Se evaluó el desempeño del sistema para videos a distintas horas**.

Se le presentaron los videos capturados al sistema ajustado y entrenado, en donde el sistema determinó las cantidades de cada vehículo que pasaron por el canal vehicular, dichas cantidades se compararon con los que realmente pasaron (conteo visual).

5. **Se establecieron las conclusiones y recomendaciones**.

Conclusiones del Diagnóstico

Una estructura compuesta de tres bloques: detector de movimiento, extractor de características y clasificador/contador de vehículos es adecuado para el diseño del Sistema para la Determinación del Flujo Vehicular basado en el Procesamiento de Imágenes y Redes Neuronales.

Se empleó el lenguaje C++ Builder ya que se logra obtener un tiempo de ejecución más corto que si se emplea Visual Basic.

Fase II Factibilidad

Una vez obtenida la información de la fase diagnóstica, se determinó la factibilidad tanto técnica y económica del proyecto. El proyecto es técnicamente factible porque existe la bibliografía adecuada y la colaboración de expertos en áreas afines a la temática del proyecto, que brindarán asesoría en la investigación y el

diseño. De igual manera se considera factible económicamente, ya que la inversión económica a realizar se encuentra dentro de un rango admisible para el autor del proyecto.

CAPITULO IV
PROPUESTA DEL ESTUDIO

Justificación

La identificación de objeto en secuencias de imágenes, es una actividad que está en pleno crecimiento en las aplicaciones donde se requiere que el sistema pueda tomar decisiones en tiempo real. En el caso del control de tráfico vehicular, se puede conocer como se está estudiando el empleo de módulos inteligentes en los circuitos cerrados de video de las principales vías, y así determinar accidentes, medir el flujo vehicular y tomar decisiones en tiempo real para solventar los problemas de tránsito.

Por lo tanto, el diseño de un sistema que determine el flujo en un canal vehicular no soluciona todos los problemas de tránsito, pero si es el comienzo de un largo camino para la automatización del control de tráfico vehicular y su aceptación por los ingenieros de tráfico.

Objetivo General

Diseñar un sistema que cuente los tipos de vehículos que transitan por un canal vehicular, empleando procesamiento de imágenes y redes neuronales.

Objetivos Específicos

- Evaluar la estructura compuesta por un detector de movimiento, un extractor de características y un clasificador/contador de vehículos en la implementación del sistema, para determinar el flujo en un canal vehicular.

- Evaluar si el tiempo de procesamiento del modelo de interacción lateral en computación acumulativa, es adecuado para extraer los objetos en movimiento de una secuencia de imágenes.

- Evaluar si el modelo de substracción de fondo junto con filtros de imágenes, es adecuado para extraer los objetos en movimiento.

- Evaluar el uso de la red de Kohonen para la extracción de características de los objetos en movimiento.

- Clasificar los objetos en movimiento por medio de una red neuronal.

- Contar los tipos de vehículos que transitan sobre una zona en una secuencia de imágenes capturadas.

Descripción de la Propuesta

La estructura del sistema para la determinación de flujo vehicular fue descrita en los alcances y se muestra en la figura 1. A continuación se describe más detalladamente cada bloque.

Detector de movimiento

La revisión teórica de las técnicas para la detección de objetos en movimientos en escenas de video, indica que el ***Modelo de Interacción Lateral en Computación Acumulativa*** puede ser una buena alternativa para implementar el detector de movimiento. Este modelo, en teoría soluciona los principales problemas en la detección de movimiento en una secuencia de imágenes y además utiliza paradigmas de la inteligencia artificial. Pero esta técnica genera un gran esfuerzo computacional al ser implementado con un computador personal que es el recurso disponible para el desarrollo del sistema, y como es esta su principal desventaja es necesario evaluar el tiempo que requiere para su procesamiento.

Por otra parte está la técnica de substracción de fondo, donde éste se adapta con el tiempo, por lo que se propone utilizar la computación acumulativa para determinar la estabilidad de cada pixel, asumiéndose cada pixel como fondo cuando permanezca estable cierta cantidad de veces. Adicionalmente, es necesario emplear filtros para disminuir el ruido causado por el dispositivo de captura. Se puede observar que teóricamente la técnica de Fernández es superior, ya que ésta supera los inconvenientes de la técnica de substracción de fondo, pero los falsos movimientos detectados por las variaciones de iluminación, movimientos de las ramas y las sombras, pueden ser solventados por los siguientes bloques que conforman al sistema.

La Figura 10 muestra un diagrama que resume el procesamiento del detector de movimiento.

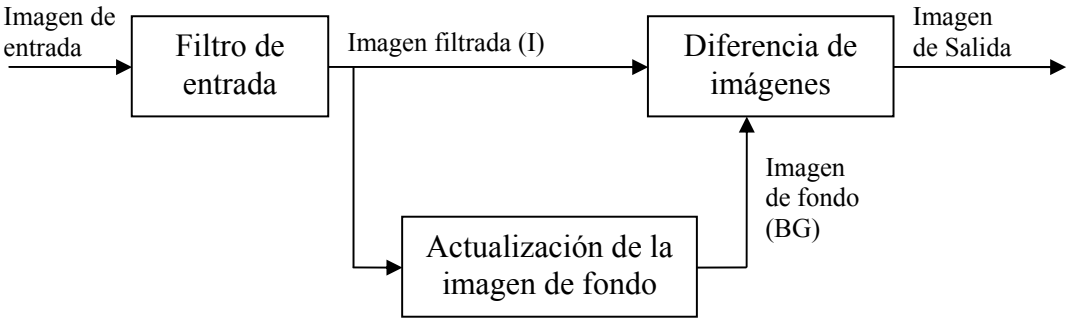

Figura 10. Diagrama del detector de movimiento

El Filtro de Entrada es el encargado de acondicionar la señal de video capturada. Se empleó un filtro gaussiano que permite realizar un suavizado de la imagen y una conversión de la imagen a escala de grises, para disminuir el ruido causado por el sensor de captura de video. La Diferencia de Imágenes se encarga de realizar un filtrado de la señal, en la que sólo deja pasar los pixeles cuya diferencia con la imagen de fondo sea superior a un umbral (*UI*).

El Bloque de Actualización de la imagen de fondo es el encargado de generar el fondo adecuado en cada instante de tiempo, permitiendo que el bloque de diferencia de la imagen sólo deje pasar los objetos en movimiento. La figura 11 muestra el diagrama del algoritmo empleado, para cada pixel de la imagen.

En el algoritmo se puede observar que cuando el sistema se inicia se genera la imagen de fondo inicial como la primera imagen capturada, luego el sistema aprende la imagen de fondo adaptando cada pixel dependiendo la permanencia de la estabilidad de éste. Para esto se contabiliza las veces que el pixel ha permanecido constante (se considera que un pixel ha permanecido constante siempre que la diferencia de dos imágenes consecutivas sea menor a un umbral (*UI*)), luego cuando

éste supere el umbral de estabilidad (U_{est}) el pixel de la imagen de fondo se iguala a la imagen actual. La imagen de fondo se actualiza solo cuando supera el umbral de estabilidad (U_{est}), esto ocurre para evitar que las regiones de los objetos homogéneas sean consideradas como imagen de fondo. Para actualizar la imagen de fondo se utiliza la matriz de cambio del pixel *M(pixel)*.

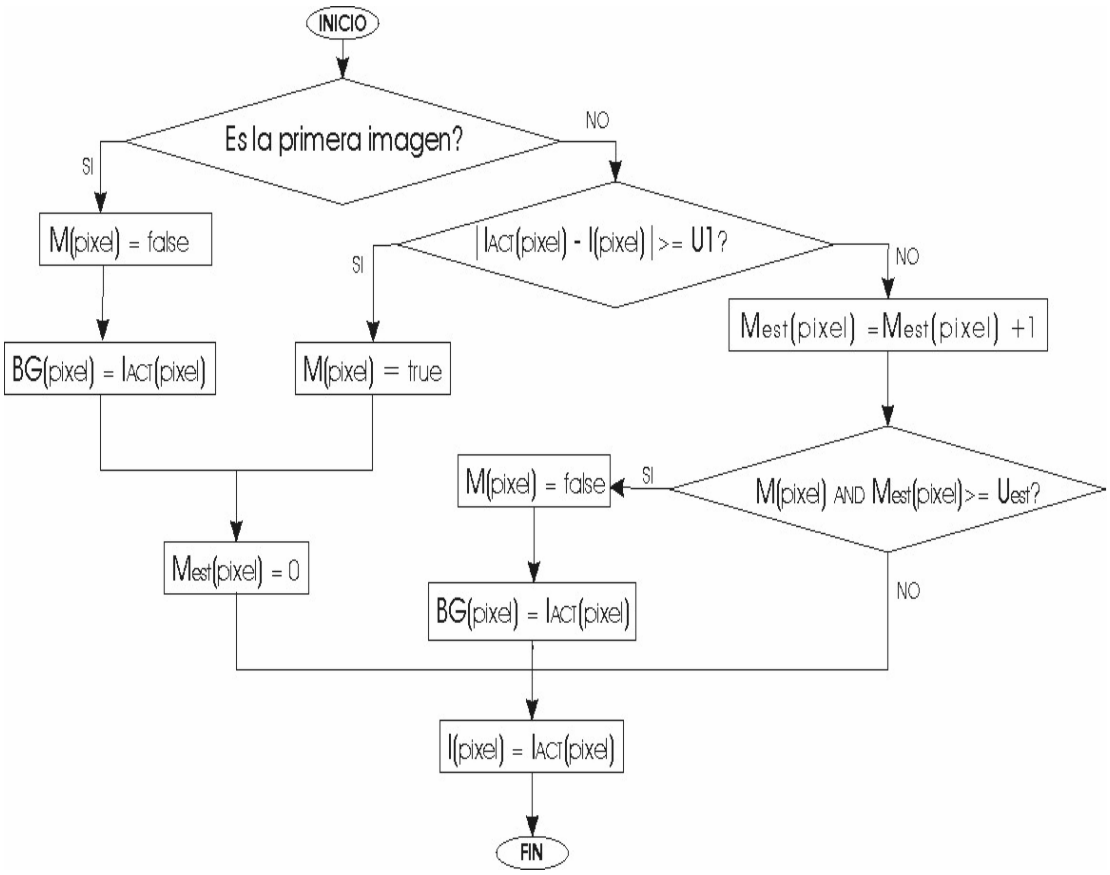

Figura 11. Algoritmo para cada pixel de la imagen del bloque Actualizar Imagen de fondo

En resumen, se propone evaluar si los requerimientos de tiempo del ***Modelo de Interacción Lateral en Computación Acumulativa*** son los adecuados para la

implementación del bloque Detector de Movimiento**,** de resultar los tiempos de procesamientos elevados se seleccionará la técnica de sustracción de fondo explicada anteriormente.

Extractor de Características

El objetivo es determinar los rasgos más significativos de los objetos en movimiento ya que al tener los pixeles característicos es más sencillo clasificar los objetos, así el clasificador de objeto tendrá una cantidad menor de datos a procesar permitiendo reducir el tiempo de procesamiento.

De la revisión teórica previa se observa que filtro de Sobel es empleado para determinar los rasgos de los objetos, ya que los contornos no se ven afectados por la variación de la intensidad de luz. Este filtro extrae los bordes de los objetos en movimiento. La cantidad de pixeles extraídos es grande y variable de un objeto a otro.

Uno de los problemas principales a enfrentar en el clasificador es que éste requiere una cantidad de pixeles constante de un objeto a otro para poder realizar la comparación con la base de conocimiento, por lo que es necesario determinar una cantidad de pixeles fijo que representen los bordes del objeto. En este caso se puede implementar un filtro de partículas como solución, el cual consiste en determinar aleatoriamente los pixeles iniciales donde a cada pixel se le suma la información que aportan sus vecinos (la cantidad de vecinos que pertenecen al borde), si superan un umbral se considera como pixel característico de lo contrario se determina uno nuevo aleatoriamente. El problema de emplear el filtro de partículas es que los pixeles extraídos pueden obviar los pixeles que mejor representen al objeto; esto quiere decir, que si aplicamos un filtro de partículas a un objeto en la misma secuencia varias

veces encontramos que los pixeles extraídos no son iguales, ya que depende de lo aleatorio del algoritmo, lo que lleva a descartar el empleo del filtro de partículas.

Por otra parte, si se emplea una red de Kohonen que se entrene para cada objeto en movimiento donde los pixeles del borde del objeto se usen como patrones de entrada, el resultado del entrenamiento es que cada neurona representa una característica principal del objeto. La estructura de la red de Kohonen seleccionada se muestra en la figura 12, donde cada neurona consta de dos pesos (las coordenadas x-y del pixel). Se selecciona una matriz de 4x4 por la geometría de los objetos a clasificar.

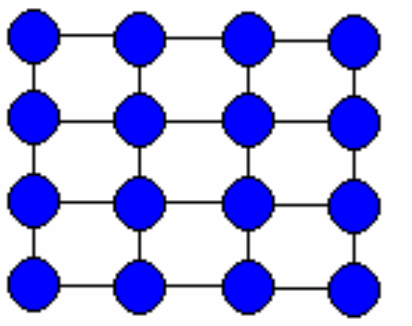

Figura 12. Estructura de la red de Kohonen

Hay que tomar en cuenta que el filtro de Sobel se aplica a toda la imagen y ésta puede contener más de un objeto en movimiento, por lo que se tiene que determinar la región de la imagen a la que pertenece cada objeto para entrenar la red de Kohonen sólo con los pixeles que pertenecen a ese objeto. Al bloque de extracción de características se le adiciona un identificador de regiones entre el filtro de Sobel y la red de Kohonen, como se muestra en la figura 13.

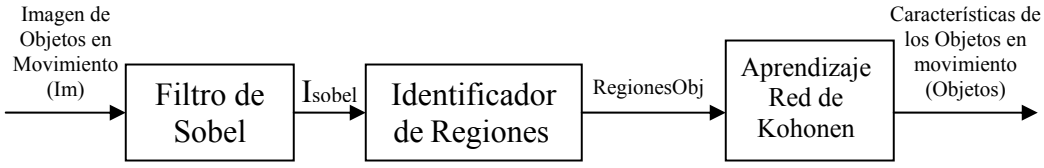

Figura 13. Diagrama del Extractor de Características

El objetivo del identificador de regiones es determinar los pixeles que pertenecen al borde de un mismo objeto, para que la red de Kohonen pueda extraer las características de cada objeto en la imagen. El algoritmo consiste en asignar a cada pixel un valor de grupo que esta directamente relacionado con el valor de grupo que tienen asignado los pixeles vecinos.

El valor de grupo al que pertenece un grupo se determina encontrando el menor valor de grupo asignado a sus vecinos. Dado el caso que los vecinos no tienen valor de grupo asignado significa que el pixel pertenece a un nuevo grupo, por lo que se incrementa el número de grupo encontrados y éste valor es el grupo al que pertenece el pixel. Una vez encontrado el valor de grupo es propagado a sus vecinos, es decir, a todos los vecinos se le asigna el mismo valor de grupo, para lograr homogeneidad.

Una vez encontrado el grupo al que pertenece cada pixel podría observarse pixeles con diferentes valores de grupo asignado perteneciente a un mismo objeto, por lo que es necesario repetir el proceso para lograr uniformidad de grupo en los objetos. Si luego de repetir el proceso de uniformidad aun se observan pixeles con diferentes valores de grupo pertenecientes a un mismo objeto se tiene que incrementar la vecindad, hasta encontrar el valor adecuado.

Clasificador/ contador de vehículos

El último bloque del sistema tiene como objetivo determinar si el objeto es un vehículo e incrementar el contador respectivo. Para lograrlo se emplea una red neuronal que clasifica el objeto en movimiento (00 no es un vehículo, 01 vehículo tipo 1 y 10 vehículo tipo 2), que luego se utiliza para incrementar los contadores respectivos de la zona que delimita al canal. Esta zona simula un detector magnético de lazo tradicional, empleado para contar los vehículos por el canal, por lo que es un área rectangular sobre un canal vehicular.

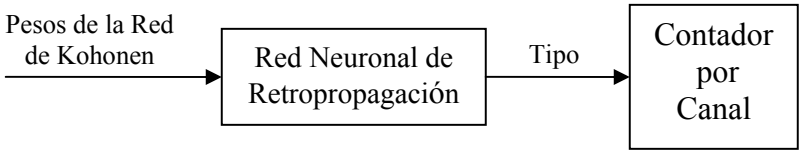

Figura 14. Diagrama del Clasificador/ contador de vehículos

La figura 14 muestra el diagrama empleado para este bloque, en el que se puede observar que la entrada del Clasificador / contador de vehículos son los pesos de la red de Kohonen que se empleó en el extractor de características, pero referido a la coordenada del objeto en movimiento, esto se cumple para que las características no dependan de la ubicación del objeto en la imagen (generalizar las características). Por lo tanto, es necesario determinar la coordenada del objeto en movimiento para referenciar los pesos de la red de Kohonen, para esto se utilizó el promedio de los pesos para cada eje (x e y).

El número de entradas que necesita la red de retropropagación está dado por el número de neuronas de la red de Kohonen multiplicado por los pesos de cada

neurona, lo que da un total de 32 (16 neuronas en la red de Kohonen x 2 pesos por neurona). Las neuronas de salidas necesarias están indicadas por los tipos de vehículos que se identificarán en el sistema, en este caso se identifican dos tipos de vehículos, por lo que las neuronas de salida necesarias son 2. Para finalizar la estructura de la red se necesita determinar la cantidad de neuronas ocultas, esto se realizó de manera iterativa comenzando con 5 y aumentando hasta que el entrenamiento de la red neuronal nos brinde una solución satisfactoria.

Para el entrenamiento de la red es necesario generar los patrones de entrenamiento, lo que se logró almacenando los pesos de la red de Kohonen del extractor de características e indicando el tipo de objeto (00 no es un vehículo, 01 es un vehículo tipo 1 y 10 para el vehículo tipo 2). Luego estos patrones son empleados para el entrenamiento utilizando el algoritmo explicado en el capitulo II, donde se comenzó con valores de 0.9 y 0.45 para η y α respectivamente, y luego se disminuyó progresivamente al observar que el error oscilaba o permanecía constante. Si la solución encontrada no es satisfactoria se incrementa el número de neuronas ocultas y se repite el procedimiento hasta encontrar la mejor solución.

Una vez clasificados los objetos en movimiento sólo hace falta incrementar los contadores para cada tipo de vehículos en un intervalo de tiempo, y así cumplir el objetivo de este trabajo implementando el algoritmo que se muestra en la figura 15.

El algoritmo primero filtra los objetos cuya localización no se encuentre en la zona de detección, luego si en la zona de detección no hay objetos se asume que el último objeto (*Xmayor*) en la zona se encuentra saliendo; es decir, el *X menor* de la zona de detección ya que los vehículos se mueven de derecha a izquierda. El valor *Xmayor* es importante ya que permite no contar los objetos que ya se contaron, pero que aún no han abandonado la zona de detección. La lógica que se emplea es simple, los objetos en un canal vehicular se mueven en un sentido (para este caso de derecha a izquierda) por lo tanto, todos los vehículos en la siguiente imagen que se encuentre a la izquierda de *Xmayor* ya se contaron por lo que se obvian.

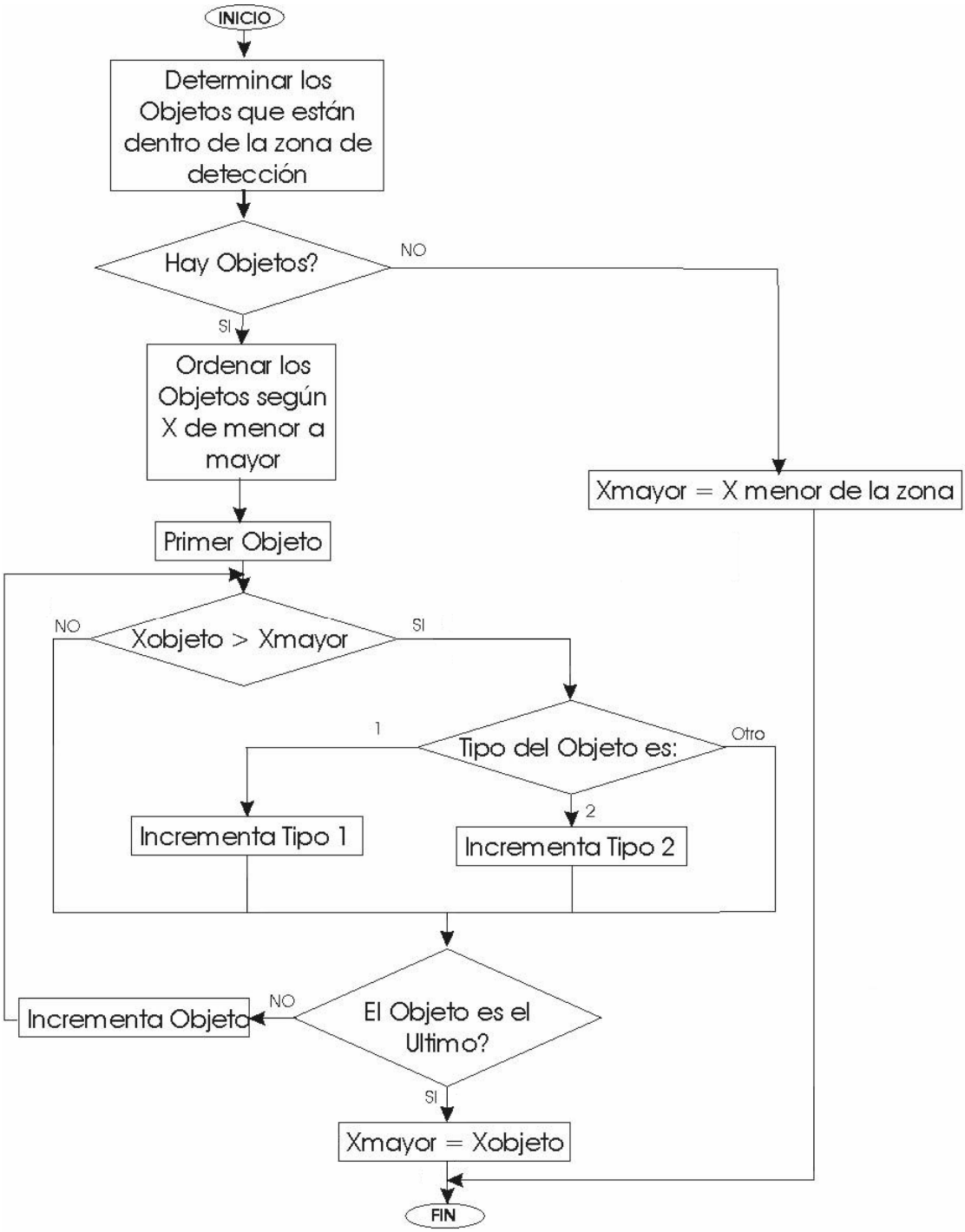

Figura 15. Algoritmo del contador de vehículos

Por otra parte si existen objetos en la zona de detección se ordenan los objetos según su localización de izquierda a derecha, para luego ir comparando en ese orden la localización de cada objeto con la ubicación del último objeto detectado en la imagen anterior, esto se ejecuta hasta que se encuentren objetos donde su localización sea mayor y así se incremente el contador del tipo respectivo. Al finalizar *Xmayor* toma el valor de la localización del último objeto detectado.

CAPITULO V
EJECUCIÓN Y EVALUACIÓN DE LA PROPUESTA

Se procedió a desarrollar cada uno de los bloques especificados en el Capítulo IV, e integrarlos; para ello, se desarrollaron y probaron los algoritmos descritos previamente.

Para realizar la evaluación del sistema fue necesario ajustarlo, lo cual se logró variando y comparando los efectos de los umbrales o parámetros de cada bloque de sistema. Se seleccionó el valor que mejor cumplió el objetivo del umbral o parámetro. A continuación se muestran los resultados obtenidos en el ajuste del sistema.

Ajuste del umbral *U1 y U2*

Los umbrales U1 y U2 deben tener el mismo valor, ya que cumplen el mismo propósito, el primero para aplicar la sustracción de fondo y el segundo para generar el fondo, ambos permiten determinar cuándo realmente un pixel se puede considerar que cambió. Para la selección de este valor se realizó la comparación del resultado de variar estos umbrales, escogiéndose el valor 15 ya que su resultado fue una imagen libre de regiones donde no ocurrieron los cambios.

Ajuste del umbral *Uest*

La figura 11 muestra la imagen de fondo para diferentes umbrales de U_{est}. Para U_{est} =0 la imagen de fondo se iguala a la imagen capturada, por lo que el resultado del

detector de movimiento es el mismo que cuando se usa la técnica de sustracción de imágenes consecutivas; al aumentar U_{est} se van obviando los pixeles con cambios para la adaptación del fondo. Se puede observar que para U_{est} = 2 las regiones homogéneas de los objetos son consideradas como pixeles de fondo; en cambio cuando la U_{est} = 5 ciertas regiones homogéneas ya no son considerados como pixeles de fondo, pero todavía se observan pixeles de los objetos que se consideran como fondo, y cuando U_{est} = 10 ya se puede observar una imagen de fondo aceptable. Hay que tomar en cuenta que U_{est} no se puede aumentar mucho, ya que es importante que el sistema adapte el fondo a los cambios de iluminación, por lo que U_{est} = 10 es una buena alternativa por los resultados obtenidos.

Figura 16. Imagen de Fondo generada para diferentes umbrales de permanencia. (a) U_{est} = 2, (b) U_{est} = 5 y (c) U_{est} = 10.

La figura 17 muestra la imagen de entrada al módulo de detector y la salida de dicho bloque, en el que se puede observar que se satisfacen los objetivos planteados para este bloque.

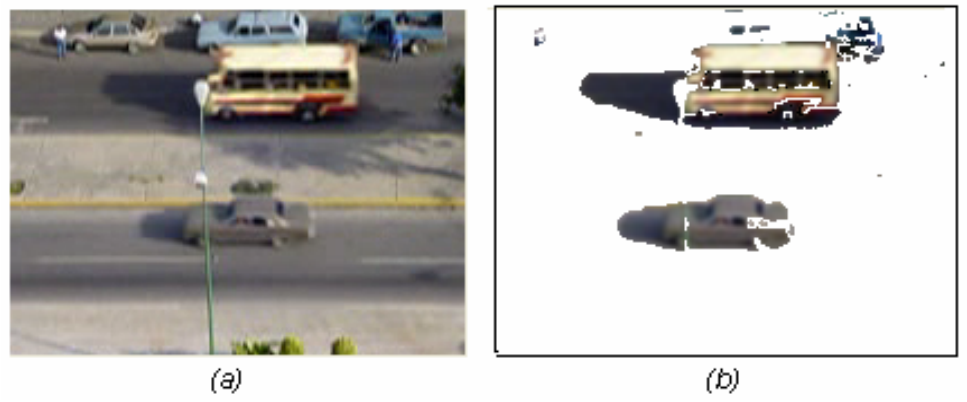

Figura 17. (a) Imagen de Entrada y (b) Imagen de salida del detector del movimiento.

Extractor de bordes

Como se mencionó en el capítulo anterior, se empleó el filtro de Sobel para la extracción de los bordes. La librería de procesamiento de video VideoLab versión 2.1 (Mitov, 2005) que contiene la implementación de dicho filtro se empleó para este prototipo. La figura 18 muestra la imagen que contiene los Objetos en Movimiento (*Im*) y el resultado al aplicar el filtro de Sobel Horizontal y Vertical (*Isobel*).

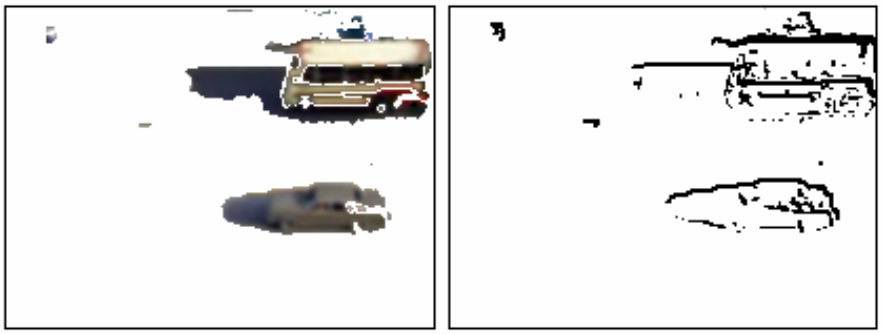

Figura 18. Imagen con los objetos en movimiento y la respuesta del filtro de Sobel de izquierda a derecha.

Ajuste del umbral U_{vecino}

La selección de la vecindad del pixel es importante para la determinación de los grupos, ya que ésta define si dos pixeles pertenecen a un mismo objeto. En la figura 19 se muestra las regiones obtenidas para diferentes valores de vecindad, en la que se puede observar que para valores de U_{vecino} menor a 3 el objeto es dividido en más de una región, lo que no es adecuado para el sistema. En cambio para una vecindad (U_{vecino}) igual a 3, cada región coincide con un objeto, por lo que éste es el valor adecuado para el sistema. También se puede observar que si se incrementa el valor de U_{vecino} se agrupa más de un objeto por región.

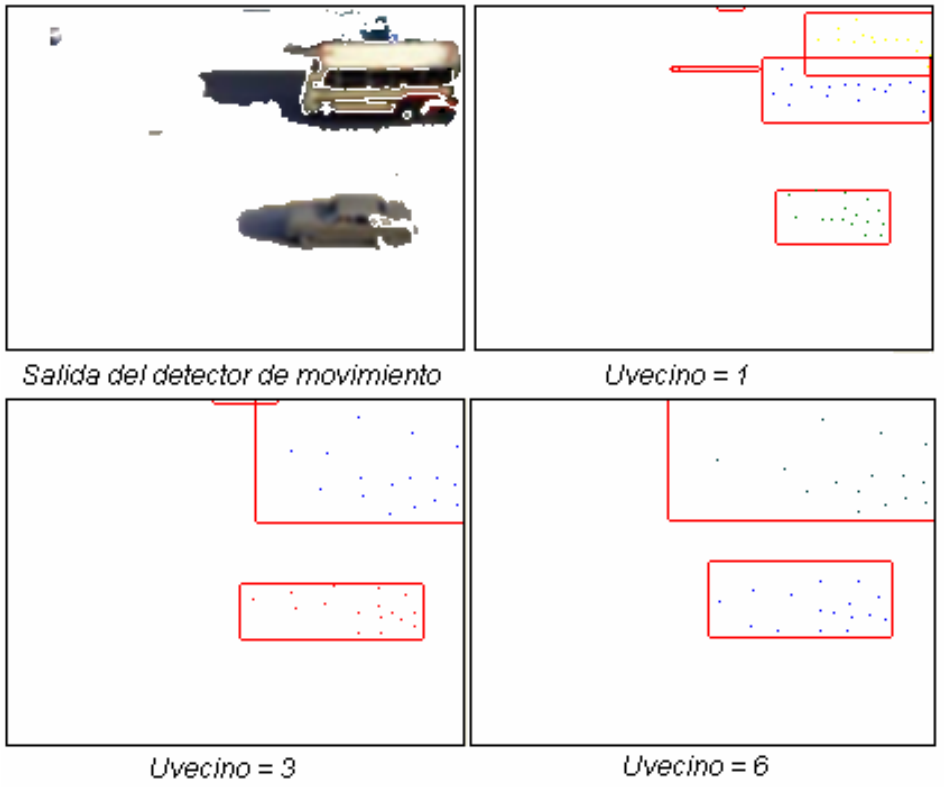

Figura 19. Regiones obtenidas para diferente valores de vecindad (U_{vecino})

Red de Kohonen

La estructura de la red de Kohonen es una matriz de 4x4 donde cada neurona representa una característica específica del objeto en movimiento. Como cada neurona representa una característica específica, se tiene que tomar en cuenta lo siguiente: Si se inician las neuronas de manera aleatoria para el entrenamiento el resultado es que las neuronas van a representar características aleatorias. Pero si la inicialización se realiza distribuyendo uniformemente las neuronas en la región de movimiento detectada, el resultado es que cada neurona representa siempre la misma característica, que es lo que se desea para el sistema.

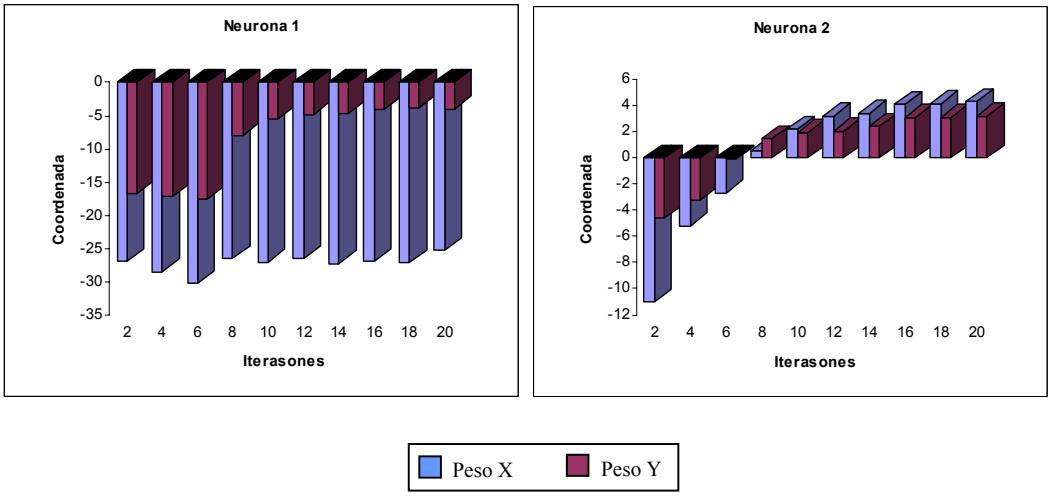

Figura 20. Coordenadas de las neuronas 1 y 2 para diferentes valores de iteraciones en el entrenamiento de la Red de Kohonen.

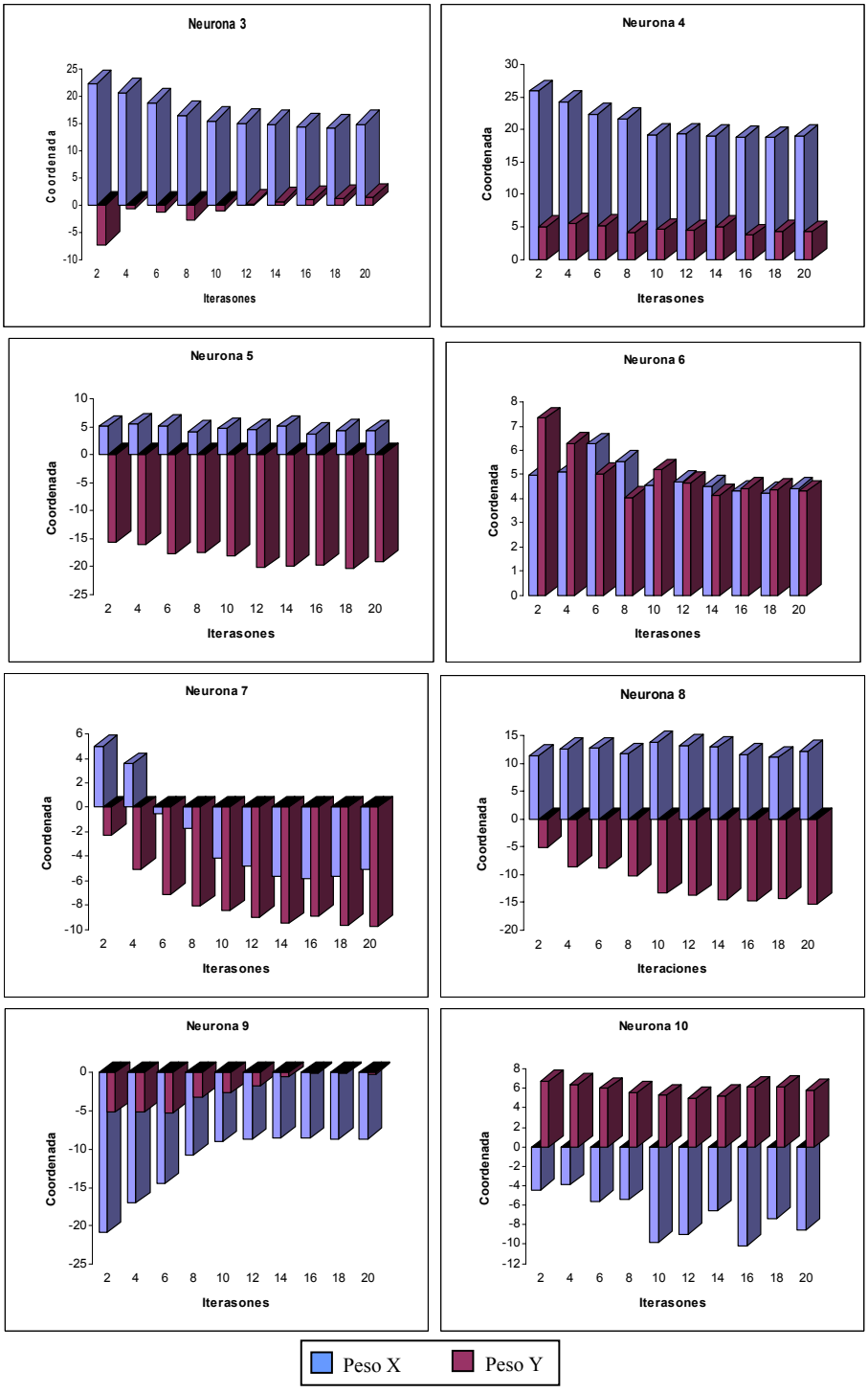

Figura 21. Coordenadas de las neuronas 2 hasta 10 para diferentes valores de iteraciones en el entrenamiento de la Red de Kohonen.

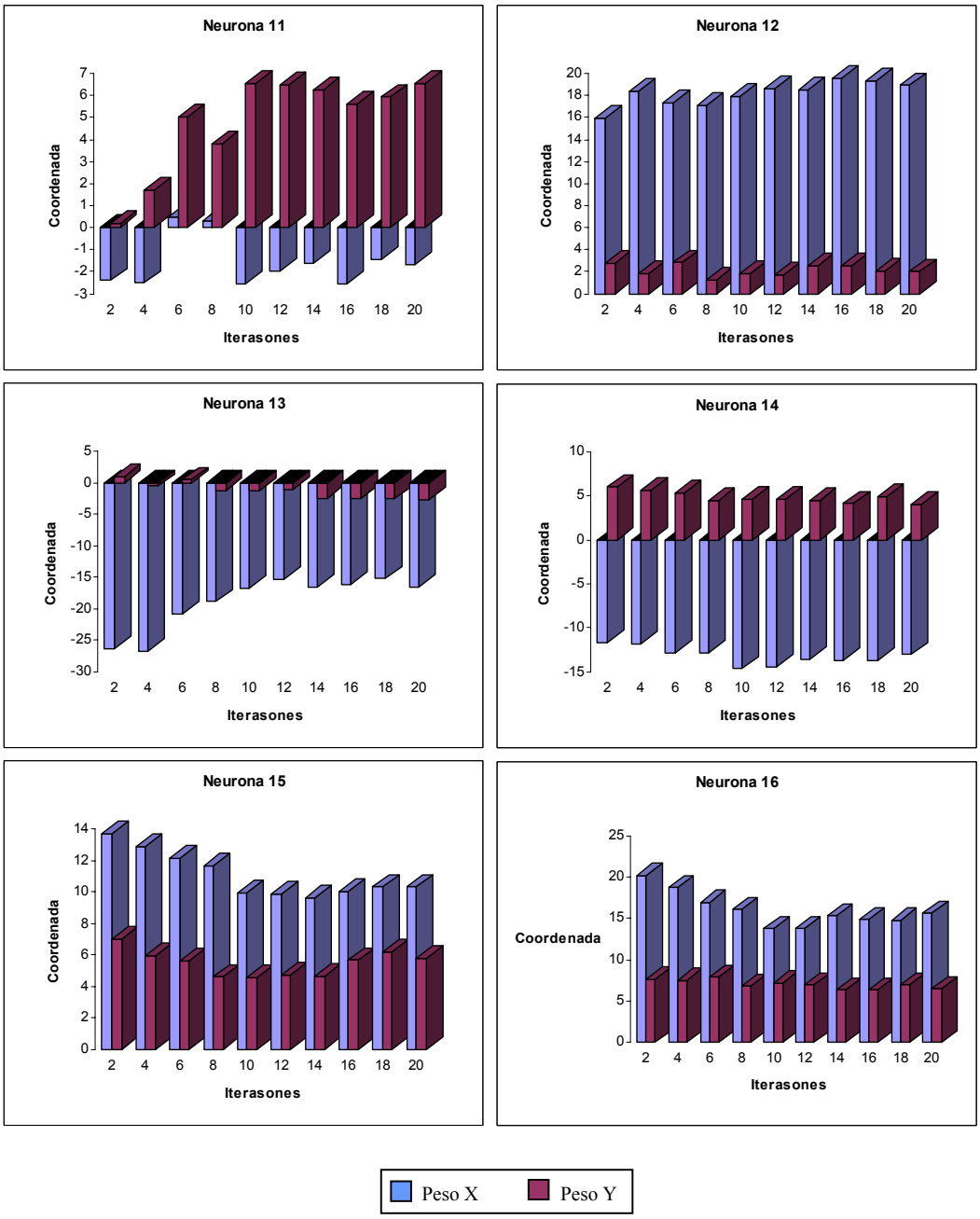

Figura 22. Coordenadas de las neuronas 11 hasta 16 para diferentes valores de iteraciones en el entrenamiento de la Red de Kohonen.

Para el entrenamiento se empleo el algoritmo explicado en las bases teóricas y en este se observa que se tiene que determinar el número de iteraciones adecuado, que tiene que ser el menor posible para reducir el costo computacional y que permita consistencia en el resultado, esto es que las características para objetos similares deben ser similares. Por lo tanto se realizó la extracción de las características de un objeto en la misma escena para diferente número de iteraciones, estas características se muestran en las figuras 20, 21 y 22. Se puede observar que para un número de iteraciones mayor a 10 las coordenadas de los pesos se mantienen similares para cada neurona. Por lo que 10 iteraciones es un buen valor para la implementación del sistema.

En la figura 23 se muestran las características extraídas de tres vehículos diferentes que pertenecen al tipo1 y en la figura 24 se muestran las características para tres vehículos diferentes del tipo 2. En ambas figuras se observa la similitud de los vehículos comparados para ciertos pixeles, lo que permite agruparlos como vehículos del mismo tipo. Además se puede observar que el centro de masa divide al vehiculo en el centro para los vehículos tipo 1, ya que los puntos mas lejanos del centro de masa son simétricos y en cambio para los vehículos del tipo 2 los pixeles característicos se encuentran concentrados a un lado del vehículo, lo que causa que el centro de masa no divida al gráfico simétricamente. También se puede observar diferencia entre los pixeles característicos de vehículos de distintos tipos, lo cual permite clasificarlos. Por lo tanto, se puede concluir que con las características extraídas por la red de Kohonen pueden ser clasificados los vehículos.

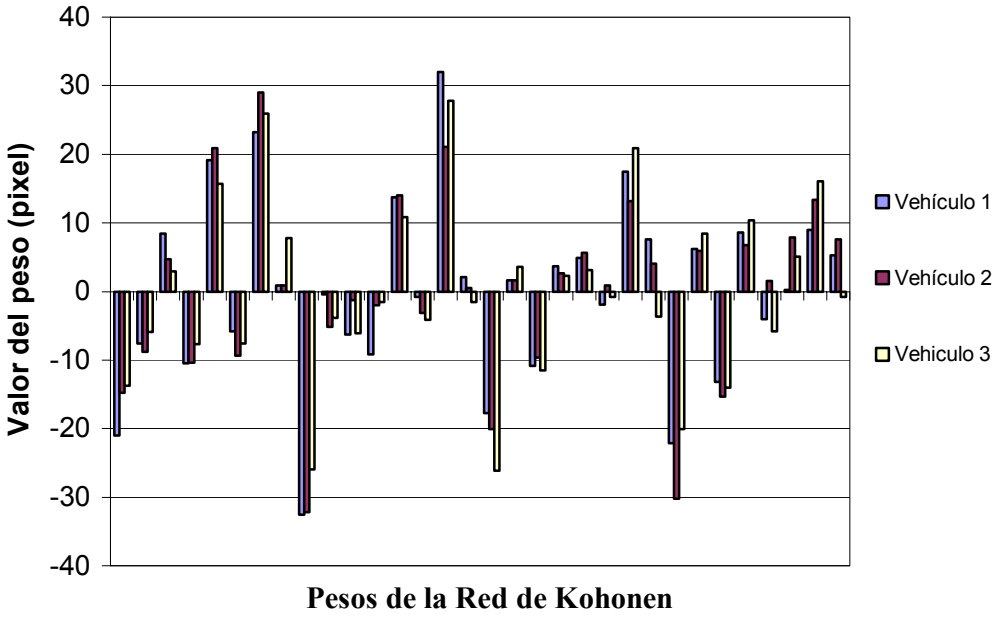

Figura 23. Pesos de la red de Kohonen para tres vehículos del tipo 1.

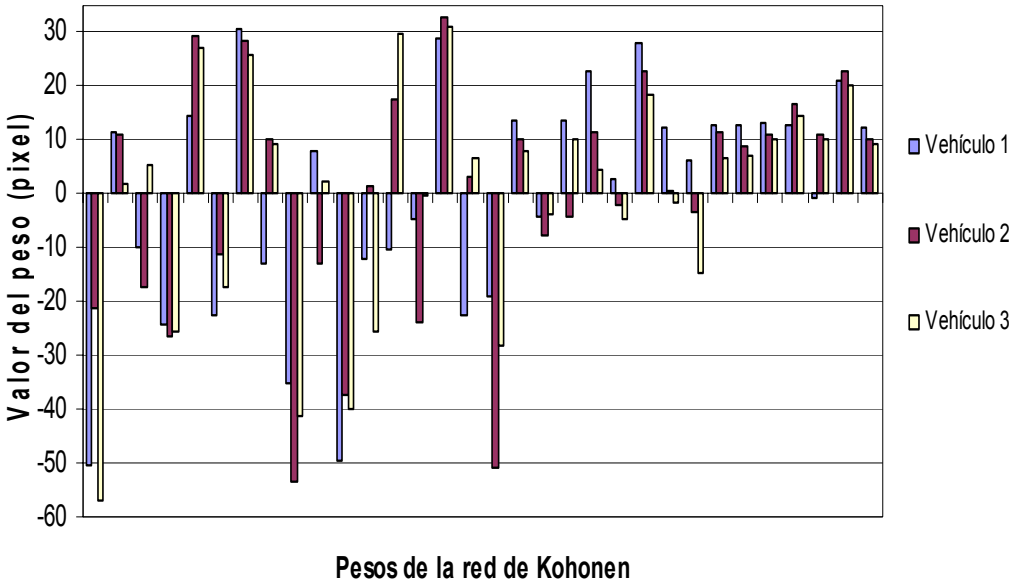

Figura 24. Pesos de la red de Kohonen para tres vehículos del tipo 2.

Red de retropropagación

El ajuste de la red de retropropagación consiste en determinar los pesos y el número de neuronas en la capa oculta que minimicen el error. Por ello, es necesario realizar el entrenamiento de la red para diferentes valores de neuronas ocultas; además es importante recordar que el entrenamiento de la red puede caer en un mínimo local, por lo que el entrenamiento para cada valor de neuronas ocultas se repitió 10 veces, donde los pesos se inicializan de manera aleatoria.

Se comenzó aplicando el algoritmo de aprendizaje como se explicó en el capítulo anterior, encontrándose un error cuadrado mínimo para los siguientes valores de neuronas ocultas:

- Para 7 neuronas ocultas se encontró un error cuadrado mínimo de 9,029.
- Para 8 neuronas ocultas se encontró un error cuadrado mínimo de 5.032.
- Para 9 neuronas ocultas se encontró un error cuadrado mínimo de 3.989.
- Para 10 neuronas ocultas se encontró un error cuadrado mínimo de 1,803.
- Para 11 neuronas ocultas se encontró un error cuadrado mínimo de 0,002.
- Para 12 neuronas ocultas se encontró un error cuadrado mínimo de 0,0018.

De los resultados anteriores se puede seleccionar 11 neuronas ocultas para la implementación del sistema, ya que seleccionar 12 neuronas ocultas sólo permite mejorar el error en 0.0002, lo que no amerita el aumento de la neurona tomando en

cuenta que el sistema requiere respuesta en tiempo real, y cualquier ahorro de costo computacional es importante. El la figura 25 muestra el error cuadrado en función del tiempo en el entrenamiento para 11 neuronas ocultas, los pesos de las neuronas ocultas se muestran en la tabla 4 y los pesos de las neuronas de salida en la tabla 5.

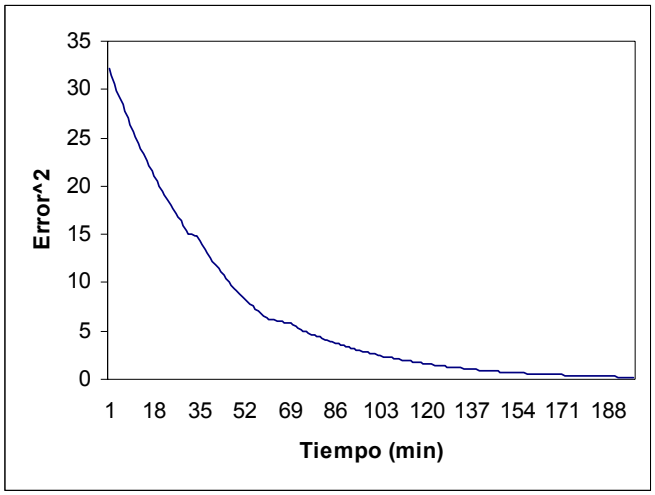

Figura 25. Error cuadrado versus tiempo en el entrenamiento de la red de retropropagación.

Tabla 4.

Pesos de las neuronas de salidas (V_{kj}) de la red de retropropagación

j	Θ_j	V_{1j}	V_{2j}
1	1.6	1.2	0.4
2	1.6	-1.9333	-4.62638
3	-1.4	1.65337	-1.1724
4	-1.8	1.84287	0.419922
5	1.2	-0.148001	-3.61416
6	-1.8	5.29773	-1.70379
7	-1.2	2.64977	-1.38607
8	0	-1.78035	-3.24889
9	0	4.41068	-0.136115
10	0	-2.90254	-0.00869656
11	0	-5.22068	-0.0155394

Tabla 5.

Pesos de las neuronas de ocultas (W_{ij}) de la red de retropropagación

j / i	1	2	3	4	5	6	7	8	9	10	11
0	1.8	1.2	0.2	-0.8	1.2	-1.4	-1.8	0	0	0	0
1	0.6	0.068667	-0.18583	1.7999	3.29334	1.39003	-1.42739	-1.21915	0.391869	-0.19951	-0.08843
2	2	-1.50245	-0.69792	-1.19983	-1.52461	0.560952	0.87358	-0.82168	0.569859	-0.30771	-0.13039
3	-1	0.996475	2.04777	1.99992	1.7677	-1.59615	1.11988	-0.30894	0.486668	-0.29738	0.03151
4	-0	-1.6984	1.07192	1.00009	1.256	0.523905	-2.04406	-0.82246	0.536547	-0.42552	0.006955
5	-0	1.13816	1.75175	0.799838	0.38052	0.662694	1.80539	0.468742	-0.23376	0.233227	0.155784
6	0.2	-0.91549	-1.09452	1.40007	-0.57131	2.11504	1.8799	-0.71889	0.162834	-0.08682	-0.18437
7	1	2.14489	-1.21507	-1.20014	-0.00086	-1.26565	0.583988	0.728731	0.665903	-0.35675	-0.2662
8	-2	2.44878	0.254142	1.59995	0.197413	-0.06683	1.90361	-0.31573	0.033128	0.00789	-0.10667
9	1.2	-3.2589	0.596529	-1.79985	-0.37503	-1.43439	0.342592	-0.89544	-0.09959	0.034067	0.155918
10	2	-0.42904	-1.94054	-1.99998	-0.11634	1.73247	0.857291	-0.36848	0.599484	-0.4113	-0.24941
11	1.2	-1.85833	1.22761	-1.2	-0.18403	-1.65187	-0.39	-0.05938	-0.14455	0.006255	0.0261
12	-1	-1.43878	0.443251	0.80015	-0.65958	-1.87993	0.956523	-0.23809	0.137071	-0.22945	-0.07531
13	1.4	0.06785	1.20915	-1.59997	-1.82431	-0.9578	1.1466	0.289861	0.049197	-0.28174	0.042549
14	0.8	0.316573	-1.16565	2.00007	0.665043	-0.21881	-1.81481	-0.18555	0.083509	-0.27184	-0.02203
15	-1	-0.71508	-1.01796	-2.00009	0.01692	-1.60208	-1.78749	1.00115	-0.10438	0.270838	-0.12348
16	1.8	-2.13915	0.78767	-0.40001	1.39792	-0.68505	-1.40359	0.068738	-0.58458	0.488307	0.109417
17	2	-0.51408	-0.18914	-1.99988	-0.77037	-1.58784	1.22722	-0.80553	0.083926	-0.0483	0.098904
18	-2	-0.30657	1.19606	1.39991	-2.33641	-0.26944	0.989324	0.215149	0.14432	-0.02734	0.098448
19	-2	-0.08784	-0.59913	1.80008	1.07774	0.051858	0.783137	-0.18402	-0.36505	0.29966	-0.00128
20	1.8	-1.27099	0.192743	0.599977	1.5213	-0.43762	1.37835	0.179171	0.313729	-0.31833	0.026455
21	0.6	2.12513	0.621532	-0.2	-0.22767	-2.00765	0.399414	0.272603	0.152902	-0.27168	-0.05157
22	-2	-0.65179	-1.4722	-1.60001	-0.81388	0.903499	0.252017	0.334981	-0.23947	0.294567	0.063128
23	-2	-0.51527	0.395259	-1.60005	-2.7154	0.929931	0.389762	0.736945	0.138775	-0.22142	0.030264
24	1.2	-0.68024	-0.44602	-1.79999	-2.53067	-1.12501	-0.92314	0.214345	-0.00775	0.053573	-0.09345
25	-1	-1.73924	1.01381	1.00017	-0.56087	-1.36417	1.88647	-0.94606	-0.29103	-0.07859	-0.14676
26	2	-1.6881	0.538426	-0.80018	1.46081	-0.05099	0.275205	0.571316	-0.95947	0.653824	0.294669
27	1	1.60768	-1.22203	0.400118	-1.93835	-1.30864	-0.15685	-0.15365	-0.5057	0.712577	0.001348
28	-1	1.24086	1.11774	-1.60009	-0.37184	-2.07817	0.074464	0.689618	-0.55243	0.400407	0.200638
29	1.6	-1.03638	0.792114	1.00004	4.98083	-1.60254	-0.58276	0.070836	-0.01311	-0.15243	-0.01624
30	-0	-0.4818	-1.48397	-0.60008	-1.66794	1.45818	1.14093	0.662766	-0.24957	0.205974	0.103465
31	-2	0.376241	-1.02635	0.599914	0.079852	-1.05573	0.260036	1.00332	-0.21206	0.351178	0.151578
32	-1	-1.20344	-0.50112	-2.00005	1.29409	-1.2822	-1.5956	0.53478	0.012786	-0.02623	-0.04153

Obtenidos los valores de ajuste, se implementó un prototipo que permitió realizar el conteo de los vehículos en videos almacenadas en el computador personal. La figura 26 muestra la interfaz del prototipo que permite al usuario ver el video, la zona de detección (región rectangular resaltada en blanco sobre el video) y los contadores por tipo de vehículos que han transitado por la zona de detección, y adicionalmente le permite:

- *Abrir videos almacenados en el computador personal*, seleccionar con el ratón en el botón de acceso rápido "Abrir", para que se despliegue la ventana de abrir archivo típica de Windows.

- *Establecer la zona de detección*, con el ratón se selecciona desde el vértice superior izquierdo al vértice inferior derecho de la zona, luego al presionar el botón "Zona" se establece la nueva zona de detección.

- *Iniciar el conteo de vehículos en el video abierto,* seleccionar con el ratón el botón rápido "Iniciar".

- *Pausar la ejecución del sistema,* seleccionar con el ratón el botón de acceso rápido "Pausa"

Al prototipo se le presentaron videos capturados en distintas horas del día, para determinar el comportamiento del sistema para diferentes intensidades de iluminación, dividiéndose los resultados en tres escenarios: videos nocturnos, videos al atardecer o al amanecer y videos en el día.

En los videos nocturnos (con poca iluminación) el sensor de captura no permitió distinguir los objetos en movimiento, ya que el sensor de captura de la cámara empleada no es especial para condiciones con poca iluminación. En estos videos se distinguen sólo las luces, lo que no permitió clasificar los tipos de vehículos para crear los patrones. En este caso sólo se puede entrenar el sistema para que cuente los vehículos sin clasificarlos.

En los videos al atardecer o al amanecer en estos existe iluminación que permite divisar los objetos en movimiento. En estos videos se observo que la cantidad de vehículos contados por el sistema y los observados coincidieron, por lo que se puede decir que se obtuvo un 100% de aciertos. En esta condición de iluminación es necesario disminuir los umbrales U1 y U2, ya que lo vehículos oscuros no tiene un cambio significativo entre el valor de un pixel perteneciente al vehiculo o el pixel que

pertenece al fondo por lo que si no se diminuyen U1 y U2 puede ocasionar que los vehículos oscuros (generalmente negros) no sean contados.

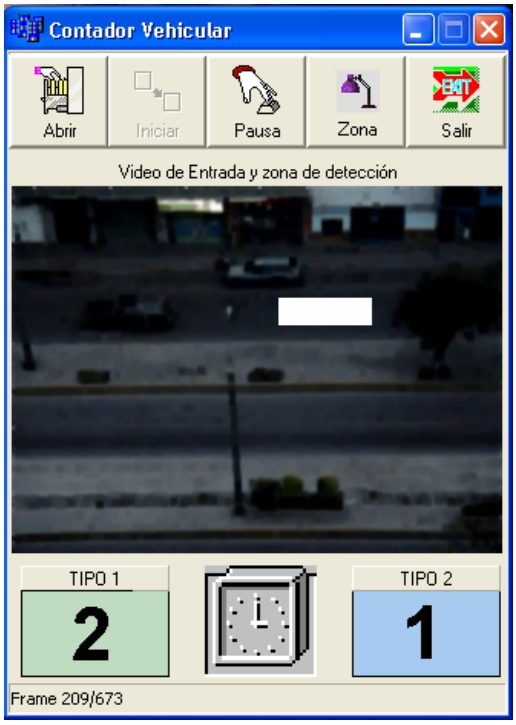

Figura 26. Ventana de Interfaz del prototipo

En los videos de día las condiciones de iluminación fueron ideales para el sensor de video empleado, donde los vehículos se definen claramente. En estos videos los vehículos contados por el sistema y los observados fueron idénticos, por lo que se obtuvo un 100% de aciertos.

CAPITULO VI
CONCLUSIONES Y RECOMENDACIONES

A través de la investigación realizada se diseñó un sistema basado en procesamiento de imágenes y redes neuronales que permitió la determinación del flujo vehicular en un canal de tránsito automotor, que cumplió con los objetivos generales y específicos.

El empleo del método de substracción de fondo en la detección de movimiento en secuencia de imágenes permitió aumentar la rapidez de respuesta del sistema, ya que centra todo el esfuerzo computacional sobre pequeñas regiones, donde se encuentran los objetos de interés.

La extracción de las características principales hace más sencillo y rápido el proceso de clasificación, ya que reduce el número de rasgos a comparar.

El modelo de interacción lateral en computación acumulativa, aún cuando teóricamente muestra ser más adecuada para la detección de objetos en video, requiere de un gran esfuerzo computacional que genera un aumento en el tiempo de respuesta del sistema, razón suficiente para no ser empleada en la implementación del mismo en nuestro caso. En cambio el método de substracción de fondo, donde el fondo se adapta utilizando computación acumulativa, permite determinar las regiones en movimiento sin emplear un esfuerzo computacional alto, mejorando satisfactoriamente el tiempo de respuesta del sistema.

La propiedad de agrupamiento de la red de Kohonen permitió extraer los pixeles característicos de los objetos en movimiento sin importar las dimensiones del objeto, y establecer un número de pixeles fijo, adecuado para la red de retropropagación que emplea estos pixeles para la clasificación.

El sistema identificó los vehículos aún cuando el recurso utilizado no haya tenido la mejor calidad gráfica y resolución. Esto permite concluir que los resultados no son alterados por estas deficiencias y que el sistema opera correctamente con un sensor de video de baja resolución, cuando hay buena iluminación.

Para solventar los problemas de solapamiento de vehículos se recomienda ubicar la cámara o sensor de captura de video sobre el canal vehicular, y definir la región de detección en la que no ocurran los solapamientos de vehículos.

Para que el sistema pueda funcionar correctamente en la detección e identificación de vehículos con poca iluminación, es necesario utilizar un sensor de captura adecuado.

Durante la investigación surgieron diferentes ideas que se podrían considerar para estudios futuros, que se enumeran a continuación:

- En la generación de los pixeles característicos de los objetos, se referencia cada peso de la red de Kohonen al punto promedio de los pesos para cada coordenada. Esta referencia puede causar un error en la identificación del objeto, ya que el movimiento de un peso a causa del ruido en la imagen o de una sombra, hace que todos los pixeles muestren un error. Si se observa que los objetos a clasificar tienen simetría se puede usar esta característica para la referencia, por lo que cada peso se puede referenciar a la distancia perpendicular a los ejes de simetría. Adicionalmente esta referencia nos permite clasificar los vehículos sin importar la inclinación que estos tengan en la imagen.

- Ensayar con otros tipos de redes de aprendizaje no supervisado, y comparar su desempeño en la extracción de características, para ver si se obtienen

mejores resultados.

- En lugar de identificar los vehículos por una zona pre-establecida que define el canal, se podría realizar el seguimiento del objeto en la imagen e identificar el canal al que pertenece según su trayectoria, empleando redes neuronales.

REFERENCIAS BIBLIOGRAFICAS

Bogomolov Y., Dror G., Lapchev S., Rivlin E. y Rudzsky M. 2003.*Classification of Moving Targets Based on Motion and Appearance*. British Machine Vision Conference (BMVC03)

Bowden, R. y KaewTraKulPong, P. 2001. *An Improved Adaptive Background Mixture Model for Real-time Tracking with Shadow Detection*. 2nd European Workshop on Advanced Video-based Surveillance Systems. Kingston upon Thames.

Canny J. 1986. *A computational approach to edge detection*. IEEE Trans. on PAMI, 8(6):679–698.

Carl, C., Goss,W., Hansen, G., Hilbert, E., Johnston, A. Y Olsasky, M. 1978. *Wide Area Detection System, Conceptual Design Study.* Jet Propulsion Lab, California. Inst. Of Tech, Pasadena. Interim Report for Federal Highway Administration, Washington, D.C.

Cheung, S. y Kamath, C. 2003. *Robust techniques for background subtraction in urban traffic video*. London, Springer - Verlag.

Cooper, L., Neskovic P. y Schuster, D. 2003. *Biologically inspired recognition system for car detection from real-time video streams*. Neural Information Processing: Research and Development, J. C. Rajapakse and L. Wang (eds.), Springer - Verlag.

Davis, L., Harwood, D. y Horprasert, T. 1999. *A statistical approach for real-time robust background subtraction and shadow detection*. IEEE ICCV'99 FRAME-RATE WORKSHOP.

Durga, P. 1999. *An Integrated Video Sensor Design for Traffic Management and Control*. World Scientific and Engineering Press, pp. 176-185

Fernández, A. 2001. *Modelos De Interacción Lateral En Computación Acumulativa Para La Obtención De Siluetas*. Universidad de Castilla-La Mancha. Tesis doctoral.

Gonzales, R. y Woods R. 1993. *Digital ImageProcessing*. Addison-Wesley Publishing Company.

Heeger D. 1987. *Model for the extraction of image flow*. Journal of the Optical Society of America, A, Vol. 4, No. 8, pp. 1455-1471.

Hinton, H., Rumelhart, D. y Williams, R. 1986. *Learning internal representations by error propagation*. Parallel Distributed Processing, volume 1. MIT Press, Cambridge, MA6.

Horn B. y Schunck B. 1981. *Determining optical flow.* Artificial Intelligence, 17, pp. 185-203.

Hue, C., Le Cadre, J. y Pérez, P. 2002. *Tracking Multiple Objects with Particle Filtering*. IEEE Transactions on Aerospace and Electronic Systems, 38(3):791-812.

Klette R., McIvor, A. y Zang, V. 2001. *The background subtraction problem for video surveillance systems*. In International Workshop Robot Vision 2001, Auckland, New Zealand, pp. 176–183.

Kollnig, H. y Nagel H. 1996. *Matching object models to segments from an optical flow field*. Fourth European Conference on Computer Vision, 15(2).

Jiménez, R., Montaño, J. y Palmer, A. 2001. *Tutorial sobre Redes Neuronales Artificiales: El Perceptrón Multicapa*. Revista Electrónica De Psicología. Vol. 5, No. 2, ISSN 1137-8492

Jiménez, R., Montaño, J. y Palmer, A. 2002. *Tutorial sobre Redes Neuronales Artificiales: Los Mapas Autoorganizados de Kohonen*. Revista Electrónica De Psicología. Vol. 6, No. 1, ISSN 1137-8492

Jones, M y Viola, P. 2001. *Rapid Object Detection using a Boosted Cascade of Simple Features*. IEEE Conference on Computer Vision and Pattern Recognition.

Michalopoulos, P. 1991. *Vehicle Detection Through Video Image Processing: The AUTOSCOPE System*. IEEE Transactions on Vehicular Technology, Vol. 40, No. 1, pp. 21-29.

Mitov, B. 2005. *VideoLab 2.1*. URL: www.mitov.com (consulta: enero 10, 2006)

P.G. 2005. *600 cámaras delatarán al culpable en los accidentes*. URL: www. 20minutos.es (consulta: marzo 1, 2006)

BIBLIOGRAFÍA

Adelson E. y Bergen J. 1985. *Spatiotemporal energy models for the perception of motion*. Journal of the Optical Society of America. Vol. 2, No. 2, pp. 284-299.

Alvarez, J., Barro, S., Fernandez, M., López, M., Manjares, A. y Mira, J. 1995. *Local accumulation of persistent activity at synaptic level: Application to motion analysis*. From Natural to Artificial Neural Computation, IWANN'95, pp. 137-143. Springer-Verlag, Germany.

Ángel, H. y Haag, M. 2000. *Incremental recognition of traffic situations from video image sequences*. Image and Vision Computing, 137(18),.

Belhumeur, P. y Hager, G. 1996. *Real-time tracking of image regions with changes in geometry and illumination*. In Proc. IEEE Conference on Computer Vision and Pattern Recognition, San Francisco, CA, pp. 403–410.

Beymer, D., Coifman, B., Malik, J. y McLauchlan, P. 1997. *A Real-Time Computer Vision System for Measuring Traffic Parameters*. Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition, San Juan, Puerto Rico, pp 495-501.

Beymer D., Coifman, B., Malik, J. y McLauchlan, P. 1998. *A Real Time Computer Vision System for vehicle tracking and traffic surveillance.* Transportation Research C 6C (4), 271-288.

Biilthof, H., Little, J. y Poggio T. 1989. *A parallel algorithm for real-time computation of optical flow*. Nature, 337:549-553.

Bogomolov Y., Dror G., Lapchev S., Rivlin E. y Rudzsky M. 2003.*Classification of Moving Targets Based on Motion and Appearance*. British Machine Vision Conference (BMVC03)

Boynton, G., Demb, J., Heeger, D., Newsome, W. y Seidemann, E., 1999. *Motion Opponency in Visual Cortex*. The Journal of Neuroscience, August 15, 19(16):7162–7174

Brandt A y Karmann K. 1990. *Moving object recognition using and adaptive background memory*. Time-Varying Image Processing and Moving Object Recognition, V. Cappellini, ed., 2, pp. 289-307

Bretzner, L. y Lindeberg, T. 1998. *Feature tracking with automatic selection of spatial scales*. Computer Vision and Image Understanding. vol. 71, pp. 385-392.

Bultho, H., Lee, S. y Poggio, T. 2000. *Biologically motivated computer vision*. Berlin: Springer-Verlag.

Camus T. 1997. *Real-Time Quantized Optical Flow*. Journal of Real-Time Imaging. Volume 3, pp. 71-86.

Carlson, B. 1997. *Clearing the Congestion: Vision Makes Traffic Control Intelligent Advanced Imaging*. URL: www.imagesensing.com (consulta: junio 10, 2005)

Cooper, L., Davis P., y Neskovic P. 2000. *Interactive parts model: an application to recognition of online cursive script*. NIPS, pp. 974–980.

Cooper , L. y Neskovic, P. 2000. *Neural network-based context driven recognition of on-line cursive script*. In 7th International Workshop on Frontiers in Handwriting Recognition, pp. 352–362.

Cooper , L. y Neskovic, P. 2001. *Object segmentation using an array of interconnected neural networks with local receptive fields*. IEEE. 0-7803-7044-9/01. pp. 1983–1988.

Craig A., Durga P. y Panos M. 1998. *A New Integrated Video Sensor Technology For Traffic Management Applications*. Presented at the Eighth Annual Meeting of ITS America, and published in the proceedings of the meeting.

Daniilidis, K., Koller, D. y Nagel, H. 1993. *Model-based Object Tracking in Monocular Image Sequences of Road Traffic Scenes*. International Journal of Computer Vision, 10: 257-281.

Davis L., Elgammal A. y Harwood D. 1999. *Non-parametric model for background subtraction.* Proceedings of IEEE ICCV'99 Frame-rate workshop.

Davis, L., Harwood, D. y Haritaoglu, I. 2000. *Real-time surveillance of people and their activities*. IEEE Transactions on Pattern Analysis & Machine Intelligence. 22(8): pp. 809-30.

Fernández, M. 1997. *Una arquitectura modular de inspiración biológica con capacidad de aprendizaje para el análisis de movimiento en secuencias de imagen en tiempo real*. Tesis Doctorales, 48. Universidad de Castilla-La Mancha. Ph. D. thesis.

Fernández A., Fernández, M., Gómez F. y Montoya, M. 2000. *Un Enfoque Neuronal al Seguimiento de un Objeto a través de su Tamaño y su Localización*. Computación y Sistemas Vol. 4 No.1 pp. 5 -17.

Fernandez A., Fernandez M., Montoya M. y Gomez F. 1999. *Tamaño y localización de un objeto: Una aproximación neuronal.* Memorias del IV Simposio Iberoamericano de Reconocimiento de Patrones SIARP'99, La Habana (Cuba), pp. 443-451.

Fernandez A., Fernandez M., Montoya M. y Moreno J. 1999. *Maximum line segments for object's motion estimation.* Proceedings of the Eighth Turkish Symposium on Artificial Intelligence and Neural Networks TAINN'99, Estambul (Turquía), pp. 145-153.

Fernandez, A., Lozano, M. y Pons A. 1998. *La inspiración biológica de los modelos computacionales de análisis de movimiento de imágenes*. Ensayos, 12, Universidad de Castilla-La Mancha.

Fernandez, M. y Mira, J. 1992. *Permanence memory: A system for real time motion analysis in image sequences*. IAPR Workshop on Machine Vision Applications, 249-252. MVA'92, Tokyo.

Fleet, D. 2000. *Design and Use of Linear Models for Image Motion Analysis*. International Journal of Computer Vision 36(3), pp. 171–193.

Fujiyoshi, H., Lipton, A. y Patil R. 1998. *Moving target classification and tracking from real-time video*. In IEEE Workshop on Applications of Computer Vision (WACV), pp. 8–14.

Fukui, K., Izumi, Y., Misu, T., Naemura, M. y Zheng, W. 2002. *Robust Tracking of Soccer Players Based on Data Fusion.* IEEE 16th International Conference on Pattern Recognition, vol.1, pp. 556-561

Grimson, W. y Stauffer, C. 1999. *Adaptive background mixture models for real-time tracking*. IEEE Computer Society Conference on Computer Vision and Pattern Recognitio). Part Vol. 2.

Grimson, W y Stauffer, C. 2000. *Learning patterns of activity using real-time tracking*. IEEE Transactions on Pattern Analysis & Machine Intelligence. 22(8): pp. 747-57.

Huang, T., Koller, D., Malik, J., Ogasawara, G., Rao, B., Russell, S. y Weber, J. 1994. *Towards robust automatic traffic scene analysis in real-time*. ICPR, Israel, Vol 1, pp. 126-131.

Isard, M. y MacCormick, J. (2001) *BraMBLe: A Bayesian multiple-blob tracker*. International Conference on Computer Vision, pp. 34-41

Jacobs, A. 1987. *Toward a model of eye movement control in visual search*. In J.K. O.Regan & A. Levy-Schoen (Eds.), Eye movements: From physiology to cognition. North-Holland: Elsevier Science Publishers.

Kabrisky, M., Keller, J., Oxley, M. y Rogers, S. 1999. *Object Recognition Based on Human Saccadic Behaviour*. Pattern Analysis and Applications. Vol. 2. Springer-Verlag. London, pp. 257-281.

Koller, D., Malik J. y Weber, J. 1994. *Robust multiple car tracking with occlusion reasoning*. Proceedings 5th European Conference on Computer Vision, Springer-Verlag, Berlin, pp. 189-196.

McKenna, S., Raja, Y. y Shaogang, G. 1998. *Object tracking using adaptive colour mixture models.* Computer Vision - ACCV '98. Third Asian Conference on Computer Vision. Proceedings. Springer-Verlag. Part vol.1. pp. 615-22.

Ohnishi, Z., Sugie N., Takeuchi, Y. y Wang Z. 1995. *Visual Tracking System Mimicking Saccadic Movements*. Proceedings of Second Asian Conference on Computer Vision, Vol. 1, pp. 131-135.

Ong, E. y Spann, M. 1999. *Robust optical flow computation based on least-median-of-squares regression*. International Journal of Computer Vision, 31:51–82.

Potter, J. 1980. *Scene segmentation using motion information*. IEEE Trans. S.M.C., 5:390–394.

Simoncelli E. 2003. *Local Analysis of Visual Motion..To appear: The Visual Neurosciences.* Chapter 109 Editors: Leo M. Chalupa and John S. Werner MIT Press.

Sudhakar, R., Xie, X. y Zhuang, H. 1995. *Real-Time Eye Feature Tracking from a Video Image Sequence Using Kalman Filter*, SMC(25), No. 12, pp. 1568-1577.

Traffic Manual. URL: http://www.wsdot.wa.gov/ (consulta: junio 10, 2005)

Zelinsky, G. 1996. *Using eye saccades to assess the selectivity of search movements*. Vision Research, 36, 2177-2187.

ANEXOS

RESUMEN CURRICULAR DEL AUTOR

ANTONIO MANUEL FERRAZ CARMONA

- Ingeniero Electrónico, Instituto Universitario Politécnico de Barquisimeto, 2003.
- Aspirante a optar al título Magíster Scientiarum en Ciencias de la Computación, Mención Inteligencia Artificial, UCLA, 2007.
- Ingeniero de Diseño en Invicta Electrónica C.A., desde el 2003 hasta el presente.

RESUMEN CURRICULAR DEL TUTOR
CARLOS IGNACIO LAMEDA MONTERO

- Ingeniero Electrónico, Instituto Universitario Politécnico de Barquisimeto, 1975.
- Master of Science en Ingeniería Eléctrica, Universidad de Stanford, USA, 1979.
- Docente del Politécnico, hoy UNEXPO, desde 1975 hasta el presente. Profesor Titular desde 1989.
- Jefe de la Sección de Control y Computación del Departamento de Ingeniería Electrónica, UNEXPO, 1980-1983, 1984-1987 y 1990-1994.
- Director Electo de Investigación y Postgrado, UNEXPO, Vicerrectorado Barquisimeto, lapsos Abril 1995-Julio1997 y Julio 1997- julio 1999.
- Profesor jubilado de la UNEXPO desde Marzo 2000.
- Diploma de Estudios Avanzados en el área de Ingeniería de Sistemas y Automática, UNED, España, 2002.
- Maestría en Ciencias de la Computación, Mención Inteligencia Artificial, UCLA, 2003.

OTROS MÉRITOS Y ACTIVIDADES:

- Presidente del Centro de Ciencias del Liceo Mario Briceño Iragorri, 1968-1969.
- Delegado Estudiantil al Consejo Académico del Politécnico, 1974.
- Diseñador de la Opción Computación en Ingeniería Electrónica del Politécnico, 1982-1985.
- Autor de 6 textos y guías de estudio a nivel universitario.
- Autor de 16 trabajos de investigación.
- Coordinador de la Unidad de Proyectos Especiales (1996-1999), la cual ha recibido los premios Innovación Tecnológica "Armada 97" (1997) y Excelencia Académica UNEXPO (1999).
- Miembro de la Junta Administradora de ENELBAR (abril 2001- mayo 2003).
- Integrante de la Comisión Regional Centroccidental del Sistema para el Reconocimiento de Méritos a los Profesores de las Universidades Nacionales (2003).
- Premios por trabajo de investigación y por trayectoria como investigador UNEXPO (1993), CONABA (1996), Orden "General Juan Jacinto Lara" (1996), CONADES (1998) y PPI (2005).
- Profesor de postgrado de la asignatura Inteligencia Artificial en la Universidad de Carabobo (2006, 2007).
- Coordinador de la Maestría en Ingeniería Electrónica de la UNEXPO (Marzo 2003 – Marzo 2007).

CÓDIGO FUENTE DEL PROTOTIPO

CÓDIGO FUENTE PARA EL ENTRENAMIENTO DE LA RED DE RETROPROPAGACIÓN

Unit1.cpp

```
//---------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
#include "Unit2.h"
#include "Unit2.cpp"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
Neurona a (1,0.5) ;
double Ea;
bool primera=true;
bool abortar =false;
double n;
double alpha;
bool regre=false;
//---------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
: TForm(Owner)
{
}
//---------------------------------------------------------------------
void __fastcall TForm1::ScrollBar1Change(TObject *Sender)
{
  NETHA->Text = ScrollBar1->Position / 1000.0;
  n = ScrollBar1->Position / 1000.0;
}
//---------------------------------------------------------------------
void __fastcall TForm1::ScrollBar2Change(TObject *Sender)
{
  ALPHA->Text = ScrollBar2->Position / 1000.0;
  alpha = ScrollBar2->Position / 1000.0;
}
//---------------------------------------------------------------------
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  double val;
  n = ScrollBar1->Position / 1000.0;
  alpha = ScrollBar2->Position / 1000.0;
  NETHA->Text = n;
  ALPHA->Text = alpha;
  val = a.Ajusta(n,alpha,false,Ea);
  Ea = val = a.Ajusta(n,alpha,true,Ea);
  this->Refresh();
  Error->SimpleText= Ea;
  if (n <= 0) Timer1->Enabled =false;
}
//---------------------------------------------------------------------
void __fastcall TForm1::EntrenarClick(TObject *Sender)
{
```

```
  if (primera)
  {
    regre =false;
    Ea = a.Ajusta(1.0,0.0,false,-1);
    primera = false;
  }
  abortar =false;
  n = ScrollBar1->Position / 1000.0;
  alpha = ScrollBar2->Position / 1000.0;
  Timer1->Enabled =true;
}
//---------------------------------------------------------------------
void __fastcall TForm1::DetenerClick(TObject *Sender)
{
  abortar=true;
  Timer1->Enabled =false;
}
//---------------------------------------------------------------------
void __fastcall TForm1::GuardarClick(TObject *Sender)
{
  a.Guardar_Pesos();
}
//---------------------------------------------------------------------
void __fastcall TForm1::SpeedButton4Click(TObject *Sender)
{
  a.Modica_Pesos();
}
//---------------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  n = ScrollBar1->Position / 1000.0;
  alpha = ScrollBar2->Position / 1000.0;
  NETHA->Text = ScrollBar1->Position / 1000.0;
  ALPHA->Text = ScrollBar2->Position / 1000.0;
  nocult->Caption = Nocultas->Position;
  a.Num_Ocultas(nocult->Caption.ToInt());
}
//---------------------------------------------------------------------
void __fastcall TForm1::NocultasChange(TObject *Sender)
{
  nocult->Caption = Nocultas->Position;
  a.Num_Ocultas(nocult->Caption.ToInt());
}
//---------------------------------------------------------------------
void __fastcall TForm1::Label3DblClick(TObject *Sender)
{
  a.CargaPesos();
}
//---------------------------------------------------------------------
```

Unit2.cpp

```
//---------------------------------------------------------------------
#pragma hdrstop
#include "Unit2.h"
#include <vcl.h>
#pragma package(smart_init)
#include <iostream.h>
#include <fstream.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <ctype.h>
#include <stdio.h>
#include <float.h>
//---------------------------------------------------------------------
class Neurona
{
 #define ENTRADA 33;   //Numero de Entradas (16*2=32)+1
 #define OCULTA 18;    //Numero de neuronas ocultas+1
 #define SALIDA 3;     //Numero de neuronas de salida+1
 int Nocultas;
 double rWji [18][33]; //Matriz de pesos de la neuronas oculta
 double rVkj[3][18];   //Matriz de pesos de las neuronas de salida
 double rdWji[18][33]; //Matriz del delta para el aprendizaje
                       //de las neuro. ocultas
 double rdVkj[3][18];  //Matriz del delta para el aprendizaje
                       //de las neuronas de salida
 double Wji [18][33];  //Matriz de pesos de la neuronas oculta
 double Vkj[3][18];    //Matriz de pesos de las neuronas de salida
 double dWji[18][33];  //Matriz del delta para el aprendizaje
                      //de las neuro. ocultas
 double dVkj[3][18];   //Matriz del delta para el aprendizaje
                      //de las neuronas de salida
 double bpj[18];
 double netha;
 double alfha;
 public:
 Neurona (double n, double a); //constructor
 ~Neurona ();
 void Salida(int *Xpi,double *ypk);
 double Ajusta(double n, double a, bool regresa,double Eantes);
 void Guardar_Pesos();
 void Modica_Pesos();
 void Num_Ocultas(int n);
 void CargaPesos();
};
//---------------------------------------------------------------------
Neurona::Neurona (double n= 0.1, double a= 0.07)
{
 Nocultas = 7;
 alfha = a;
 netha = n;
 randomize();
 for (int j=0 ;j<Nocultas;j++)
 {
   for (int i=0; i<33;i++)
   {
```

```
      do
      {
        Wji[j][i]= 2.0*random (21)/10.0 - 2.0;
      } while (Wji[j][i]==0);
      dWji[j][i]=0;
    }
    for (int k=0; k<3;k++)
    {
      do
      {
        Vkj[k][j] = 2.0*random (21)/10.0 - 2.0;
      } while (Vkj[k][j]==0);
      dVkj[k][j]=0;
    }
  }
}
//---------------------------------------------------------------------
Neurona::~Neurona (void)
{
}
//---------------------------------------------------------------------
void Neurona::Salida(int *Xpi, double *Ypk)
{
  // se calcula la suma
  double suma =0;
  //se calcula las salidas de la capa oculta
  for (int j=1;j<9;j++)
  {
    for (int i=1;i<Nocultas-1;i++) suma += Wji[j][i] * Xpi[i];
    suma += Wji[j][0];
    bpj[j] = (1.0/(1.0+exp(-1.0*suma)));
  }
  //se calcula las salidas de la neuronas de salida
  for (int k=1;k<3;k++)
  {
    suma =0;
    for (int j=1;j<Nocultas-1;j++) suma += Vkj[k][j] * bpj[j];
    suma += Vkj[k][0];
    Ypk[k]= (1.0/(1.0+exp(-1.0*suma)));
  };
}
//---------------------------------------------------------------------
double Neurona::Ajusta(double n, double a,bool error, double Eantes)
{
  double Ypk[3];
  double Dpk[3];
  float Xpi[33];
  double dWjia[18][33]; //Matriz del delta para el aprendizaje
                        //de las neuro. ocultas
  double dVkja[3][18];  //Matriz del delta para el aprendizaje
                        //de las neuronas de salida
  double dpk[3];
  double dpj[18];
  double E=0;
  double Ep=0;
  long double sumad=0;
  long double suma =0;
```

```
double diferencia;
int i,j,k;
int Tipo=0;
netha = n;
alfha = a;
ifstream sal;
sal.open("patrones");
if (!sal) return 0.0;
E=0;
//se inicializa para la suma de todos los patrones
for (int j=0;j<Nocultas ;j++)
{
  for (int k=0;k<3;k++) dVkja[k][j]=0;
  for (int i=0;i<33;i++) dWjia[j][i]=0;
}
do
{
  //se cargan en la entrada el patrón
  char c;
  float v;
  for (int i =1 ;i<33;i++)
  {
    sal >> Xpi[i];
    sal.get(c);
  }
  sal >> Tipo;
  sal.get(c); //se lee entre o retorno del carro
  switch (Tipo)
  {
    case 1://
      Dpk[1]=1;
      Dpk[2]=0;
      break;
    case 3://
      Dpk[1]=0;
      Dpk[2]=1;
      break;
    default:
      Dpk[1]=0;
      Dpk[2]=0;
  }
  //para cada patrón
  // Se calcula la suma
  Ep=0;
  //se calcula las salidas de la capa oculta
  for (int j=1;j<Nocultas;j++)
  {
    suma =0;
    for (int i=1;i<33;i++) suma += Wji[j][i] * Xpi[i];
    suma += Wji[j][0];
    suma = -1.0 * suma;
    suma = expl(suma);
    suma = 1.0 + suma;
    suma = 1.0 / suma;
    bpj[j] = suma;
  }
  //se calcula las salidas de las neuronas de salida
  for (int k=1;k<3;k++)
```

```
    {
      suma =0;
      for (int j=1;j<Nocultas;j++)
        suma += double(Vkj[k][j]) * double(bpj[j]);
      suma += double(Vkj[k][0]);
      suma = -1.0 * suma;
      suma = expl(suma);
      suma = 1.0 + suma;
      suma = 1.0 / suma;
      Ypk[k]= suma;
      diferencia = Dpk[k] - Ypk[k];
      Ep += diferencia * diferencia;
      diferencia= diferencia * Ypk[k];
      suma = (1.0 - Ypk[k]);
      dpk[k] = diferencia * suma;
      for (int j=0;j<Nocultas;j++)
        dVkja[k][j] += dpk[k] * bpj[j]; //se calcula la suma
    };
    Ep = Ep/2.0;
    for (int j=1;j<Nocultas;j++)
    {
      sumad=0;
      for (int k=1;k<3;k++) sumad += dpk[k] * Vkj[k][j];
      sumad = sumad * (1.0 - bpj[j]);
      dpj[j] = bpj[j] * sumad;
      dWjia[j][0] += dpj[j]; // se calcula la suma
      for (int i=1;i<33;i++) dWjia[j][i] += dpj[j] * Xpi[i];
    }
    E += Ep; // se calcula el error
    //hasta aquí para cada patrón
  } while (!sal.eof());
  if (error == false)
  {
    //lo que esta a continuación va después de que se hayan mostrado
    //todos los patrones
    for (int j=0;j<Nocultas;j++)
      for (int i=1;i<33;i++)
      {
        double dantes = dWji[j][i];
        dWji[j][i]= netha * double(dWjia[j][i]) + alfha * dantes;
        Wji[j][i]+= dWji[j][i];
      }
   for (int j=0;j<Nocultas;j++)
      for (int k=1;k<3;k++)
      {
        double dantes = dVkj[k][j];
        dVkj[k][j]= netha * dVkja[k][j] + alfha * dantes;
        Vkj[k][j] += dVkj[k][j];
      }
  }
  sal.close();
  return E;
}
//---------------------------------------------------------------------
void Neurona::Guardar_Pesos()
{
  ofstream sal("PesosW");
  if (sal)
```

```
  {
    //se realiza un respaldo de los pesos
    for (int j=0;j<Nocultas ;j++)
      for (int i=0;i<33;i++)
      {
        sal << Wji [j][i];
        sal << " ";
      }
    sal.close();
    ofstream sal2("PesosV");
    if (sal2)
    {
      //se realiza un respaldo de los
      for (int j=0;j<Nocultas ;j++)
        for (int k=0;k<3;k++)
        {
          sal2 << Vkj[k][j];
          sal2 << " ";
        }
      sal2.close();
    }
  }
}
//---------------------------------------------------------------------
void Neurona::Modica_Pesos()
{
  randomize();
  for (int j=0 ;j<Nocultas;j++)
  {
    for (int i=0; i<33;i++)
      if(random (11)>2)
      {
         Wji[j][i]= 2.0*random (21)/10.0-2.0;
         dWji[j][i]=0;
       }
     for (int k=0; k<3;k++)
       if(random (11)>2)
       {
          Vkj[k][j]= 2.0*random (21)/10.0-2.0;
          dVkj[k][j]=0;
        }
    }
}
//---------------------------------------------------------------------
void Neurona::Num_Ocultas(int n)
{
  Nocultas=n;
}
//---------------------------------------------------------------------
void Neurona::CargaPesos()
{
  //se cargan los pesos
  ifstream sal1;
  char c;
  double valor;
  sal1.open ("PesosW");
  if (sal1)
  {
    //se realiza un respaldo de los
```

```
    for (int j=0;j<Nocultas ;j++)
      for (int i=0;i<33;i++)
      {
        sal1 >> valor;
        Wji[j][i] = valor;
        sal1.get(c);
      }
    sal1.close();
    ifstream sal2;
    sal2.open ("PesosV");
    if (sal2)
    {
      //se realiza un respaldo de los
      for (int j=0;j<Nocultas ;j++)
        for (int k=0;k<3;k++)
        {
          sal2 >> Vkj[k][j];
          sal2.get(c);
        }
      sal2.close();
    }
    else
      ShowMessage("No se pudo cargar los Pesos Vkj");
  }
  else
    ShowMessage("No se pudo cargar la matriz de Pesos");
}
```

CÓDIGO FUENTE DEL DETECTOR DE VEHICULOS

Unit1.cpp

```
//---------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "stdlib.h"
#include "Unit1.h"
#include "iostream.h"
#include "fstream.h"
#include <iostream.h>
#include <fstream.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <ctype.h>
#include <stdio.h>
#include <float.h>
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "VLAVIPlayer"
#pragma link "VLCommonFilter"
#pragma link "VLGenericFilter"
#pragma link "VLImageDisplay"
#pragma link "VLCommonDisplay"
#pragma link "VLGrayScale"
#pragma link "VLGammaFilter"
#pragma link "VLBoxFilter"
#pragma link "VLFixedFilter"
#pragma link "VLMotionDetect"
#pragma link "VLMorphFilter"
#pragma link "VLCombine"
#pragma link "VLMultiInput"
#pragma link "VLDSVideoPlayer"
#pragma link "OWLComps"
#pragma link "VLCommonGen"
#pragma link "VLImageGen"
#pragma resource "*.dfm"
#include "SLCRealBuffer.h"

double Wji [18][33]; //Matriz de pesos de la neuronas oculta
double Vkj[3][18]; //Matriz de pesos de las neuronas de salida

//---------------------------------------------------------------------
class CParticulaObj
{
  //public:
  int X; // coordenada x
  int Y; // coordenada y
  int Objeto; //Numero del objeto al que pertenece
  bool HayVecinos;
  struct EVecinos
  {
    int X;
    int Y;
    int d;
    bool existe;
  } Vecino[8];
```

```
  public:
  //Metodos
  void Inicializa ();
  void SetXY(int xi,int yi);
  void setObjeto(int Obj);
  void HallarVecinos (TVLCVideoDataAccess I,int w,int h);
  int DameX ();
  int DameY ();
  int DameObjeto ();
  bool VecinoExiste(int i);
  bool DameHayVecinos();
  int VecinoX(int i);
  int VecinoY(int i);
  int Vecinod(int i);
  void Pintar_Obj (TVLCVideoDataAccess I,int xo,int yo,
  int V,int w,int h,int Obj);
  int Hallar_Obj (TVLCVideoDataAccess I,int xo,int yo,
  int V,int w,int h);
};

void CParticulaObj::Pintar_Obj (TVLCVideoDataAccess I,int xo,int yo,
                                int V,int w,int h,int Obj)
{
  int xd = xo - V;
  int yd = yo - V;
  if (xd<0) xd=0;
  if (yd<0) yd=0;
  int xm = xo + V+1;
  int ym = yo + V+1;
  if (xm>w) xm=w;
  if (ym>h) ym=h;
  for (int i = xd; i<xm ;i++)
    for (int j=yd; j<ym;j++)
      if(I.GetGreen(i,j)==0) I.SetGreen(i,j,Obj);
      else
        if (Obj <I.GetGreen(i,j)) I.SetGreen(i,j,Obj);

};

int CParticulaObj::Hallar_Obj (TVLCVideoDataAccess I,int xo,int yo,
                               int V,int w,int h)
{
  int Obj=0;
  int xm = xo - V;
  int ym = yo - V;
  if (xm<0) xm=0;
  if (ym<0) ym=0;
  int xu = xo + V+1;
  int yu = yo + V+1;
  if (xu>w) xu=w;
  if (yu>h) yu=h;
  for (int i = xm; i<xu ;i++)
    for (int j=ym; j<yu;j++)
      if(I.GetGreen(i,j)>0)
        if (Obj==0) Obj = I.GetGreen(i,j);
```

```
        else
          if (Obj >I.GetGreen(i,j)) Obj= I.GetGreen(i,j);
  return Obj;
};

bool CParticulaObj::DameHayVecinos ()
{
  return HayVecinos;
}

void CParticulaObj::Inicializa ()
{
  Objeto=0;
  HayVecinos=false;
  for (int i =0 ;i <8;i++)
    Vecino[i].existe=false;
}

void CParticulaObj::SetXY (int xi,int yi)
{
  X=xi;
  Y=yi;
}

void CParticulaObj::setObjeto (int Obj)
{
  Objeto=Obj;
}

void CParticulaObj::HallarVecinos (TVLCVideoDataAccess I,int w,
                                   int h)
{
  for(int i =1; i < w-X; i++)
  {
    if (I.GetGreen(X+i,Y)>0)
    {
      HayVecinos=true;
      Vecino[0].existe=true;
      Vecino[0].X=X+i;
      Vecino[0].Y=Y;
      Vecino[0].d=i;
      break;
    }
  }
  //para el vecino 1
  for(int i =1; i < w-X && i < Y+1; i++)
    if (I.GetGreen(X+i,Y-i)>0)
    {
      HayVecinos=true;
      Vecino[1].existe=true;
      Vecino[1].X=X+i;
      Vecino[1].Y=Y-i;
      Vecino[1].d=i;
      break;
    }

  //para el vecino 2
  for(int i =1; i <Y+1 ; i++)
```

```
  if (I.GetGreen(X,Y-i)>0)
  {
    HayVecinos=true;
    Vecino[2].existe=true;
    Vecino[2].X=X;
    Vecino[2].Y=Y-i;
    Vecino[2].d=i;
    break;
  }
//para el vecino 3
for(int i =1; i <Y+1 && i < X+1 ; i++)
  if (I.GetGreen(X-i,Y-i)>0)
  {
    HayVecinos=true;
    Vecino[3].existe=true;
    Vecino[3].X=X-i;
    Vecino[3].Y=Y-i;
    Vecino[3].d=i;
    break;
  }
//para el vecino 4
for(int i =1; i < X+1 ; i++)
  if (I.GetGreen(X-i,Y)>0)
  {
    HayVecinos=true;
    Vecino[4].existe=true;
    Vecino[4].X=X-i;
   Vecino[4].Y=Y;
    Vecino[4].d=i;
    break;
  }
//para el vecino 5
for(int i =1; i < X+1 && i< h-Y ; i++)
  if (I.GetGreen(X-i,Y+i)>0)
  {
    HayVecinos=true;
    Vecino[5].existe=true;
    Vecino[5].X=X-i;
    Vecino[5].Y=Y+i;
    Vecino[5].d=i;
    break;
  }
//para el vecino 6
for(int i =1; i< h-Y ; i++)
  if (I.GetGreen(X,Y+i)>0)
  {
    HayVecinos=true;
    Vecino[6].existe=true;
    Vecino[6].X=X;
    Vecino[6].Y=Y+i;
    Vecino[6].d=i;
    break;
  }
  //para el vecino 7
  for(int i =1; i< h-Y && i<w-X ; i++)
    if (I.GetGreen(X+i,Y+i)>0)
    {
```

```
            HayVecinos=true;
            Vecino[7].existe=true;
            Vecino[7].X=X+i;
            Vecino[7].Y=Y+i;
            Vecino[7].d=i;
            break;
        }
    }
}

int CParticulaObj::DameX ()
{
  return X;
}

int CParticulaObj::DameY ()
{
  return Y;
}

int CParticulaObj::DameObjeto ()
{
  return Objeto;
}

bool CParticulaObj::VecinoExiste(int i)
{
  return Vecino[i].existe;
}

int CParticulaObj::VecinoX(int i)
{
  return Vecino[i].X;
}

int CParticulaObj::VecinoY(int i)
{
  return Vecino[i].Y;
}


int CParticulaObj::Vecinod(int i)
{
  return Vecino[i].d;
}

class Capa
{
  public:
  bool *dato;
  Capa();
  void SetSize (int x, int y, int i );
  void Inicia (int x, int y);
};

Capa::Capa ()
{
  dato = new bool [2];
```

```
};

void Capa::SetSize (int x, int y, int i )
{
  if (i==0) {delete [] dato;};
  dato = new bool [x*y];
};

void Capa::Inicia (int x, int y)
{
  for ( int x1 = 0; x1 < x; x1 ++ )
    for ( int y1 = 0; y1 < y; y1 ++ )
      dato[x1 + x*y1] = false;
};

class Capai
{
  public:
  short int *dato;
  Capai ();
  void SetSize (int x, int y, int i );
  void Inicia (int x, int y);
};

Capai::Capai ()
{
  dato = new short int [2];
};
void Capai::SetSize (int x, int y, int i )
{
  if (i==0) {delete [] dato;};
  dato = new short int [x*y];
  Inicia(x,y);
};

void Capai::Inicia (int x, int y)
{
  for ( int x1 = 0; x1 < x; x1 ++ )
    for ( int y1 = 0; y1 < y; y1 ++ )
      dato[x1 + x*y1] = 0;
};
//---------------------------------------------------------------------------
TForm1 *Form1;
//---------------------------------------------------------------------------
// variables globales
struct EPatron
{
  int x;
  int y;
} *P;
int N;
struct EWk
{
  float x;
  float y;
} Wk[16];
struct ERegion
```

```
{
  int Xmenor;
  int Xmayor;
  int Ymenor;
  int Ymayor;
  float Xobjeto;
  int Obejtos;
  int Tipo1;
  int Tipo2;
} Region;
struct EVehiculos
{
  float X;
  int Tipo;
} Vehiculos [10];
int *cPart;
int w = 0;   //Xmax
int h = 0;   //Ymax
int U1= 15;  //umbral por defecto por defecto
int Uest=30; //umbral para definir cuando un pixel se puede
             //considerar que se estabilizo
int U3= 15;  //umbral para la sustracción del fondo
int U4 = 150;
int Uvecino = 3;
int Tipo =0;
bool AlmacenaPatron=false;
int Umasa = 15;
int wParticula=2;
int hParticula=2;
int Dv=5;
int Dx= 10;
int ObjetosOb=0;
int ObjMos=1;
int Objetos=0;
long Tparticulas;
bool detener=false;
bool Estado;
bool Cambio;
int Frame=0;
Capai Mest;
Capai BG;
Capai I;
Capa M;
long NParticulas = 2000;
CParticulaObj *PartObj;
int Ocultas = 15;
//---------------------------------------------------------------------
void __fastcall TForm1::VLGenericFilter1ProcessData(TObject *Sender,
TVLCVideoBuffer InBuffer, TVLCVideoBuffer &OutBuffer,
bool &SendOutputData)
{
  //Actualización y generación del BG
  TVLCVideoDataAccess InData = InBuffer.Data();
  TVLCVideoDataAccess OutData = OutBuffer.Data();
  if (Frame ==0)
  {
```

```
    //cuando es la primera imagen se inicializan
    //los parámetros y la imagen para BG
    w = InData.GetWidth();
    h = InData.GetHeight();
    NParticulas=w*h+1;
    Estado =false;
    BG.SetSize(w,h,0);
    I.SetSize(w,h,0);
    M.SetSize(w,h,0);
    Mest.SetSize(w,h,0);
    Frame=2;
    PartObj = new CParticulaObj [NParticulas];
    P = new EPatron [NParticulas];
    long i=0 ;
    for ( int y = 0; y < h ; y ++ )
      for ( int x = 0; x < w; x ++ )
      {
        int Pixeli = InData.GetRed(x,y)*0.229;
        Pixeli += InData.GetGreen(x,y)*0.587;
        Pixeli += InData.GetBlue(x,y)*0.114;
        I.dato[i]= Pixeli;
        BG.dato[i]= Pixeli;
        i += 1;
       //se crea la imagen de respaldo
      };
  }
  else
  {
    long i =0;
    for ( int y = 0; y < h ; y ++ )

      for ( int x = 0; x < w; x ++ )
      {
        //se convierte a escala de gris
        Estado =true;
        int Pixeli = InData.GetRed(x,y)*0.229;
        Pixeli += InData.GetGreen(x,y)*0.587;
        Pixeli += InData.GetBlue(x,y)*0.114;
        int valaux = Pixeli - I.dato[i];
        valaux = abs( valaux );
        if (valaux >= U1)
        {
          Mest.dato[i] = 0;
          M.dato[i] = True;
        }
        else
        {
           Mest.dato[i] += 1;
           if (M.dato[i] && (Mest.dato[i] > Uest))
           {
             M.dato[i] = False;
             BG.dato[i]= Pixeli;
           }
        }
        I.dato[i] = Pixeli;
        i += 1;
      };
   }
}
```

```
//---------------------------------------------------------------------
void Inicializacion (void)
{
  Frame =0;
  Cambio = true;
}
//---------------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Region.Xmenor=173;
  Region.Xmayor=235 ;
  Region.Ymenor= 72;
  Region.Ymayor= 91;
  detener =false;
  cPart= new int [1];
  Inicializacion();
  Player->Paused =true;
  Player->FileName = Open1->FileName;
  U1Edit->Text = U1;
  U3Edit->Text = U3;
  UestEdit->Text = Uest;
  //se cargan los pesos
  ifstream sal1;
  char c;
  double valor;
  sal1.open ("PesosW");

  if (sal1)
  {
    //se realiza un respaldo de los
    for (int j=0;j<Ocultas ;j++)
      for (int i=0;i<33;i++)
      {
        sal1 >> valor;
        Wji[j][i] = valor;
        sal1.get(c);
      }
    sal1.close();
    ifstream sal2;
    sal2.open ("PesosV");
    if (sal2)
    {
      //se realiza un respaldo de los
      for (int j=0;j<Ocultas ;j++)
        for (int k=0;k<3;k++)
        {
           sal2 >> Vkj[k][j];
           sal2.get(c);
        }
      sal2.close();
    }
    else
      ShowMessage("No se pudo cargar los Pesos Vkj");
  }
  else
    ShowMessage("No se pudo cargar la matriz de Pesos");
}

void __fastcall TForm1::Button1Click(TObject *Sender)
```

```
{
  Inicializacion();
}
//---------------------------------------------------------------------
void __fastcall TForm1::Framefin(TObject *Sender, int StartFrame,
                                 int EndFrame, int CurrentFrame)
{
  String A;
  A = CurrentFrame;
  BarraEstado->SimpleText = "Frame " + A;
  A = EndFrame;
  BarraEstado->SimpleText = BarraEstado->SimpleText + "/" + A;
  if (CurrentFrame == EndFrame)
  {
    Player->Paused =true;
    Player->CurrentFrame = 5;
    Iniciar->Enabled =true;
    Pausa->Enabled =false;
    BarraEstado->SimpleText = "Video finalizado";
  }
}

//---------------------------------------------------------------------
void __fastcall TForm1::VLGenericFilter2ProcessData(TObject *Sender,
               TVLCVideoBuffer InBuffer, TVLCVideoBuffer &OutBuffer,
               bool &SendOutputData)
{
  TVLCVideoDataAccess InData = InBuffer.Data();
  TVLCVideoDataAccess OutData = OutBuffer.Data();
  if (Estado)
  {
    long i =0 ;
    for ( int y = 0; y < h ; y ++ )
      for ( int x = 0; x < w; x ++ )
      {
        int Pixeli = InData.GetRed(x,y)*0.229;
        Pixeli += InData.GetGreen(x,y)*0.587;
        Pixeli += InData.GetBlue(x,y)*0.114;
        int valaux = Pixeli;
        valaux -= BG.dato[i];
        valaux = abs(valaux);
        if (valaux < U3)
        {
          OutData.SetGreen(x,y,0);
          OutData.SetRed(x,y,0);
          OutData.SetBlue(x,y,0);
        }
        i +=1;
        //se realiza la sustracción del fondo ponderado
      };
   }
   else
   {
     for ( int y = 0; y < h ; y ++ )
       for ( int x = 0; x < w; x ++ )
         {
            OutData.SetGreen(x,y,0);
            OutData.SetRed(x,y,0);
            OutData.SetBlue(x,y,0);
```

```
          //no se muestra nada hasta que el BG este estable
        };
   }
}
//---------------------------------------------------------------------
void __fastcall TForm1::VLGenericFilter4ProcessData(TObject *Sender,
              TVLCVideoBuffer InBuffer, TVLCVideoBuffer &OutBuffer,
              bool &SendOutputData)
{
  TVLCVideoDataAccess OutDat = OutBuffer.Data();
  TVLCVideoDataAccess InData = InBuffer.Data();
  if (Estado)
  {
    for ( int x = 0 ; x < w ; x ++)
      for ( int y = 0; y < h; y ++ )
      {
        int Pixeli = InData.GetRed(x,y)*0.229;
        Pixeli += InData.GetGreen(x,y)*0.587;
        Pixeli += InData.GetBlue(x,y)*0.114;
        if (Pixeli > 5) OutDat.SetPixel(x,y,clWhite);
        else OutDat.SetPixel(x,y,clBlack);
      }
   }
}
//---------------------------------------------------------------------
void __fastcall TForm1::MostrarBGFilterProcessData(TObject *Sender,
              TVLCVideoBuffer InBuffer, TVLCVideoBuffer &OutBuffer,
              bool &SendOutputData)
{
  TVLCVideoDataAccess OutData = OutBuffer.Data();
  for ( int y = 0; y < h ; y ++ )
    for ( int x = 0; x < w; x ++ )
    if (Region.Xmayor > x && Region.Xmenor < x
        && Region.Ymayor > y && Region.Ymenor < y)
           OutData.SetPixel(x,y,clWhite);
}
//---------------------------------------------------------------------
void __fastcall TForm1::EscalaGricesHumProcessData(TObject *Sender,
              TVLCVideoBuffer InBuffer, TVLCVideoBuffer &OutBuffer,
              bool &SendOutputData)
{
  //Actualización y generación del BG
  TVLCVideoDataAccess InData = InBuffer.Data();
  TVLCVideoDataAccess OutData = OutBuffer.Data();
  int Pixeli;
  for ( int y = 0; y < h ; y ++ )
    for ( int x = 0; x < w; x ++ )
    {
      Pixeli = InData.GetRed(x,y)*0.229;
      Pixeli += InData.GetGreen(x,y)*0.587;
      Pixeli += InData.GetBlue(x,y)*0.114;
      if (InData.GetRed(x,y) > U1) OutData.SetPixel(x,y,clWhite);
      else OutData.SetPixel(x,y,0);
      if (InData.GetBlue(x,y) > U1) OutData.SetPixel(x,y,clWhite);
      else OutData.SetPixel(x,y,0);
      if (InData.GetGreen(x,y) > U1) OutData.SetPixel(x,y,clWhite);
      else OutData.SetPixel(x,y,0);
    };
```

```
}
//---------------------------------------------------------------------
long ProcesaParticulas(TVLCVideoDataAccess I)
{
  long NP =1;
  for ( int y = 0; y < h ; y ++ )
    for ( int x = 0; x < w; x ++ )
    {

      if (I.GetGreen(x,y)>0)
      {
         PartObj[NP].SetXY(x,y);// posicion de la particula
         NP++;
      }
    };
  return NP-1; //se retorna el Total de particulas Procesadas
}

int ObtenerObj(TVLCVideoDataAccess I,long TotalP)
{
  int Objetos=0;
  for (int k =0 ;k<2;k++)
    for (long CP =1;CP<TotalP+1;CP++)
    {
      int Obj = PartObj[CP].Hallar_Obj(I,PartObj[CP].DameX(),
      PartObj[CP].DameY(),
      Uvecino,w,h );
      if(Obj==0)
      {
        Objetos++;
        Obj=Objetos;
      };
      PartObj[CP].Pintar_Obj(I,PartObj[CP].DameX(),
      PartObj[CP].DameY(),
      Uvecino,w,h,Obj);
    }
   cPart= new int [Objetos+1];
   for (int i =1; i < Objetos+1; i++) cPart[i]=0;
   for (long CP =1;CP<TotalP+1;CP++)
   {
     int Obj = I.GetGreen(PartObj[CP].DameX(),
     PartObj[CP].DameY());
     cPart[Obj]++;
     PartObj[CP].setObjeto(Obj);
   }
   return Objetos;
}

void __fastcall TForm1::VLGenericFilter5ProcessData(TObject *Sender,
               TVLCVideoBuffer InBuffer, TVLCVideoBuffer &OutBuffer,
               bool &SendOutputData)
{
  EWk Wk1[16];
  Region.Obejtos=0;
  TVLCVideoDataAccess InData = InBuffer.Data();
  TVLCVideoDataAccess OutData = OutBuffer.Data();
  Tparticulas = ProcesaParticulas( InData);
  //se limpia la ventana de salida
```

```
for (int x=0;x<w;x++)
  for (int y=0;y<h;y++)
    OutData.SetPixel(x,y,0);
if (Tparticulas>0)
{
  Objetos = ObtenerObj(OutData,Tparticulas+1);
  int Obje =0;
  //Se extraen los objetos
  //se limpia la ventana de salida
  for (int x=0;x<w;x++)
    for (int y=0;y<h;y++)
      OutData.SetPixel(x,y,0);
  for (int O=1;O<Objetos+1;O++)
  {
    if (cPart[O]>64)
    {
      N=0;
      Obje++;
      int Xmenor=w+1;
      int Ymenor=h+1;
      int Xmayor=0;
      int Ymayor=0;
      for (long i =1; i<Tparticulas+1;i++)
        if (PartObj[i].DameObjeto() == O)
        {
           N++;
           int valor = PartObj[i].DameY();
           P[N].y = valor;
           if (valor > Ymayor) Ymayor = valor;
           if (valor < Ymenor) Ymenor = valor;
           valor = PartObj[i].DameX();
           P[N].x= valor;
           if (valor > Xmayor) Xmayor = valor;
           if (valor < Xmenor) Xmenor = valor;
        }
      //inicializacion del mapa de kohonen
      int dx = (Xmayor-Xmenor)/4;
      int dy = (Ymayor-Ymenor)/4;
      Wk1[0].x= Xmenor;
      Wk1[0].y=Ymenor;
      for (int i = 1; i<16;i++)
      {
        Wk1[i].x= Wk1[i-1].x +dx;
        Wk1[i].y=Wk1[i-1].y;
        if(i%4==0)
        {
          Wk1[i].x=Xmenor;
          Wk1[i].y= Wk1[i-1].y +dy;
        }
      }
      //aprendizaje y extracción de características
      //se entrena solo para 10 iteraciones
      float a=0.1; //constante de aprendizaje
      int g=0;
      for(int Tk=0; Tk < 10;Tk++)
      {
        a=0.1*(1-Tk/10.0); //contante de aprendizaje
```

```
//1º deteccion de la ganadora
for (int i =1; i<N+1;i++)
{
  g=0;
  float Dg=0;
  int Resto=0;
  int Cociente=0;
  float Da=0;
  float delta=0;
  Dg= abs(P[i].x - Wk1[0].x) + abs(P[i].y - Wk1[0].y);
  for (int k=1; k<16;k++)
  {
    Da= abs(P[i].x - Wk1[k].x) + abs(P[i].y - Wk1[k].y);
    if (Dg > Da)
    {
      Dg=Da;
      g=k;
    }
  }
  //Ajuste de W
  delta = a*(P[i].x-Wk1[g].x);
  Wk1[g].x+=delta;
  delta = a*(P[i].y-Wk1[g].y);
  Wk1[g].y+=delta;
  if (Tk<6)
  {
    //solo se ajustan los vecinos hasta una
    //determinada iteración
    Resto= g % 4;
    Cociente = g / 4;
    float aux;
    aux=a;
    a = a*0.01;
    switch (Resto)
    {
        case 0://
          switch (Cociente)
          {
            case 0://
              delta = a*(P[i].x-Wk1[g+1].x);
              Wk1[g+1].x+=delta;
              delta = a*(P[i].x-Wk1[g+4].x);
              Wk1[g+4].x+=delta;
              delta = a*(P[i].y-Wk1[g+1].y);
              Wk1[g+1].y+=delta;
              delta = a*(P[i].y-Wk1[g+4].y);
              Wk1[g+4].y+=delta;
              break;
            case 3://
              delta = a*(P[i].x-Wk1[g+1].x);
              Wk1[g+1].x+=delta;
              delta = a*(P[i].x-Wk1[g-4].x);
              Wk1[g-4].x+=delta;
              delta = a*(P[i].y-Wk1[g+1].y);
              Wk1[g+1].y+=delta;
              delta = a*(P[i].y-Wk1[g-4].y);
              Wk1[g-4].y+=delta;
              break;
            default:
```

```
      delta = a*(P[i].x-Wk1[g-4].x);
      Wk1[g-4].x+=delta;
      delta = a*(P[i].x-Wk1[g+1].x);
      Wk1[g+1].x+=delta;
      delta = a*(P[i].x-Wk1[g+4].x);
      Wk1[g+4].x+=delta;
      delta = a*(P[i].y-Wk1[g-4].y);
      Wk1[g-4].y+=delta;
      delta = a*(P[i].y-Wk1[g+1].y);
      Wk1[g+1].y+=delta;
      delta = a*(P[i].y-Wk1[g+4].y);
      Wk1[g+4].y+=delta;
  }
  break;
case 3://Nk-1
  switch (Cociente)
  {
    case 0://
      delta = a*(P[i].x-Wk1[g-1].x);
      Wk1[g-1].x+=delta;
      delta = a*(P[i].x-Wk1[g+4].x);
      Wk1[g+4].x+=delta;
      delta = a*(P[i].y-Wk1[g-1].y);
      Wk1[g-1].y+=delta;
      delta = a*(P[i].y-Wk1[g+4].y);
      Wk1[g+4].y+=delta;
      break;
    case 3://
      delta = a*(P[i].x-Wk1[g-1].x);
      Wk1[g-1].x+=delta;
      delta = a*(P[i].x-Wk1[g-4].x);
      Wk1[g-4].x+=delta;
      delta = a*(P[i].y-Wk1[g-1].y);
      Wk1[g-1].y+=delta;
      delta = a*(P[i].y-Wk1[g-4].y);
      Wk1[g-4].y+=delta;
      break;
    default:
      delta = a*(P[i].x-Wk1[g-4].x);
      Wk1[g-4].x+=delta;
      delta = a*(P[i].x-Wk1[g-1].x);
      Wk1[g-1].x+=delta;
      delta = a*(P[i].x-Wk1[g+4].x);
      Wk1[g+4].x+=delta;
      delta = a*(P[i].y-Wk1[g-4].y);
      Wk1[g-4].y+=delta;
      delta = a*(P[i].y-Wk1[g-1].y);
      Wk1[g-1].y+=delta;
      delta = a*(P[i].y-Wk1[g+4].y);
      Wk1[g+4].y+=delta;
  }
  break;
default:

switch (Cociente)
  {
    case 0://
      delta = a*(P[i].x-Wk1[g-1].x);
      Wk1[g-1].x+=delta;
      delta = a*(P[i].x-Wk1[g+1].x);
```

```
                Wk1[g+1].x+=delta;
                delta = a*(P[i].x-Wk1[g+4].x);
                Wk1[g+4].x+=delta;
                delta = a*(P[i].y-Wk1[g-1].y);
                Wk1[g-1].y+=delta;
                delta = a*(P[i].y-Wk1[g+1].y);
                Wk1[g+1].y+=delta;
                delta = a*(P[i].y-Wk1[g+4].y);
                Wk1[g+4].y+=delta;
                break;
              case 3://
                delta = a*(P[i].x-Wk1[g-1].x);
                Wk1[g-1].x+=delta;
                delta = a*(P[i].x-Wk1[g+1].x);
                Wk1[g+1].x+=delta;
                delta = a*(P[i].x-Wk1[g-4].x);
                Wk1[g-4].x+=delta;
                delta = a*(P[i].y-Wk1[g-1].y);
                Wk1[g-1].y+=delta;
                delta = a*(P[i].y-Wk1[g+1].y);
                Wk1[g+1].y+=delta;
                delta = a*(P[i].y-Wk1[g-4].y);
                Wk1[g-4].y+=delta;
                break;
              default:
                delta = a*(P[i].x-Wk1[g-4].x);
                Wk1[g-4].x+=delta;
                delta = a*(P[i].x-Wk1[g+1].x);
                Wk1[g+1].x+=delta;
                delta = a*(P[i].x-Wk1[g-1].x);
                Wk1[g-1].x+=delta;
                delta = a*(P[i].x-Wk1[g+4].x);
                Wk1[g+4].x+=delta;
                delta = a*(P[i].y-Wk1[g-4].y);
                Wk1[g-4].y+=delta;
                delta = a*(P[i].y-Wk1[g+1].y);
                Wk1[g+1].y+=delta;
                delta = a*(P[i].y-Wk1[g-1].y);
                Wk1[g-1].y+=delta;
                delta = a*(P[i].y-Wk1[g+4].y);
                Wk1[g+4].y+=delta;
            }
          }
          a=aux;
        }
      }
    }

double Ypk[3];
float Xpi[33];
double bpj[18];
long double suma =0;
float Ym=0;
float Xm=0;
for (int k=0;k<16;k++)
{
  Xm += Wk1[k].x;
  Ym += Wk1[k].y;
}
Xm=Xm/16.0;
```

```
Ym=Ym/16.0;
for (int k=0;k<16;k++)
{
  //se crea el vector de entrada
  Wk1[k].x-=Xm;
  Wk1[k].y-=Ym;
}
int i =1;
for (int k =0; k<16;k++)
{
  Xpi[i]=Wk1[k].x;
  i++;
  Xpi[i]=Wk1[k].y;
  i++;
}
//se calcula las salidas de la capa oculta
for (int j=1;j<Ocultas;j++)
{
  suma =0;
  for (int i=1;i<33;i++) suma += Wji[j][i] * Xpi[i];
  suma += Wji[j][0];
  suma = -1.0 * suma;
  suma = expl(suma);
  suma = 1.0 + suma;
  suma = 1.0 / suma;
  bpj[j] = suma;
}
//se calcula las salidas de las neuronas de salida
for (int k=1;k<3;k++)
{
  suma =0;
  for (int j=1;j<Ocultas;j++)
    suma += double(Vkj[k][j]) * double(bpj[j]);
  suma += double(Vkj[k][0]);
  suma = -1.0 * suma;
  suma = expl(suma);
  suma = 1.0 + suma;
  suma = 1.0 / suma;
  Ypk[k]= suma;
  if (Ypk[k]>0.5) Ypk[k]=1.0;
  else Ypk[k]=0.0;
};
int tipo=0;
TColor color = clBlack;
if (Ypk[1]==0.0 && Ypk[2]==1.0 )
{
  tipo = 2;
  color = clYellow;
}
if (Ypk[1]==1.0 && Ypk[2]==0.0 )
{
  tipo = 1;
  color = clBlue;
}
if (Xm < Region.Xmayor && Xm > Region.Xmenor)
{
  Region.Obejtos++;
  Vehiculos[Region.Obejtos].X = Xm;
  Vehiculos[Region.Obejtos].Tipo= tipo;
```

```
          for (int y= Ymenor+1;y<Ymayor;y++)
            for (int x= Xmenor+1;x<Xmayor;x++)
            {
              OutData.SetPixel(x,y,color);
              OutData.SetPixel(x,y,color);
            }
        }
      }
    }
    ObjetosOb=Obje;
  }
}
//---------------------------------------------------------------------
void __fastcall TForm1::VLGenericFilter6ProcessData(TObject *Sender,
               TVLCVideoBuffer InBuffer, TVLCVideoBuffer &OutBuffer,
               bool &SendOutputData)
{
  TVLCVideoDataAccess OutData = OutBuffer.Data();
  //se limpia la ventana de salida
  for (int x=0;x<w;x++)
    for (int y=0;y<h;y++)
    OutData.SetPixel(x,y,clWhite);
    if (Tparticulas>0 && Player->Paused ==true )
    {
      EWk Pk[16];
      long N;
      int ObjetoP=0;
      for (int O=1; O<Objetos+1;O++)
      {
        if (cPart[O]>64)
        {
          ObjetoP++;
          if (ObjetoP==ObjMos)
          {
            N=0;
            int Xmenor=w+1;
            int Ymenor=h+1;
            int Xmayor=0;
            int Ymayor=0;
            for (long i =1; i<Tparticulas+1;i++)
              if (PartObj[i].DameObjeto() == O)
              {
                N++;
                int valor = PartObj[i].DameY();
                P[N].y = valor;
                if (valor > Ymayor) Ymayor = valor;
                if (valor < Ymenor) Ymenor = valor;
                valor = PartObj[i].DameX();
                P[N].x= valor;
                if (valor > Xmayor) Xmayor = valor;
                if (valor < Xmenor) Xmenor = valor;
              }
            //inicialización del mapa de kohonen
            int dx = (Xmayor-Xmenor)/4;
            int dy = (Ymayor-Ymenor)/4;
            Pk[0].x= Xmenor;
            Pk[0].y=Ymenor;
            for (int i = 1; i<16;i++)
```

```
{
  Pk[i].x= Pk[i-1].x +dx;
  Pk[i].y=Pk[i-1].y;
  if(i%4==0)
  {
    Pk[i].x=Xmenor;
    Pk[i].y= Pk[i-1].y +dy;
  }
}
//aprendizaje y extracción de características
//se entrena solo para 10 iteraciones
float a=0.1; //constante de aprendizaje
int g=0;
for(int Tk=0; Tk < 10;Tk++)
{
  a=0.1*(1-Tk/10.0); //constante de aprendizaje
                     //1º detección de la ganadora
  for (int i =1; i<N+1;i++)
  {
    g=0;
    float Dg=0;
    int Resto=0;
    int Cociente=0;
    float Da=0;
    float delta=0;
    Dg=abs(P[i].x -Pk[0].x) + abs(P[i].y - Pk[0].y);
    for (int k=1; k<16;k++)
    {
      Da=abs(P[i].x-Pk[k].x)+abs(P[i].y - Pk[k].y);
      if (Dg > Da)
      {
        Dg=Da;
        g=k;
      }
    }
    //Ajuste de W
    delta = a*(P[i].x-Pk[g].x);
    Pk[g].x+=delta;
    delta = a*(P[i].y-Pk[g].y);
    Pk[g].y+=delta;
    if (Tk<6)
    {
      //solo se ajustan los vecinos hasta una
      //determinada iteración
      Resto= g % 4;
      Cociente = g / 4;
      float aux;
      aux=a;
      a=a*0.01;
      switch (Resto)
      {
            case 0://
              switch (Cociente)
              {
                case 0://
                  delta = a*(P[i].x-Pk[g+1].x);
                  Pk[g+1].x+=delta;
                 delta = a*(P[i].x-Pk[g+4].x);
```

```
        Pk[g+4].x+=delta;
        delta = a*(P[i].y-Pk[g+1].y);
        Pk[g+1].y+=delta;
        delta = a*(P[i].y-Pk[g+4].y);
        Pk[g+4].y+=delta;
        break;
      case 3://
        delta = a*(P[i].x-Pk[g+1].x);
        Pk[g+1].x+=delta;
        delta = a*(P[i].x-Pk[g-4].x);
        Pk[g-4].x+=delta;
        delta = a*(P[i].y-Pk[g+1].y);
        Pk[g+1].y+=delta;
        delta = a*(P[i].y-Pk[g-4].y);
        Pk[g-4].y+=delta;
        break;
      default:
        delta = a*(P[i].x-Pk[g-4].x);
        Pk[g-4].x+=delta;
        delta = a*(P[i].x-Pk[g+1].x);
        Pk[g+1].x+=delta;
        delta = a*(P[i].x-Pk[g+4].x);
        Pk[g+4].x+=delta;
        delta = a*(P[i].y-Pk[g-4].y);
        Pk[g-4].y+=delta;
        delta = a*(P[i].y-Pk[g+1].y);
        Pk[g+1].y+=delta;


        delta = a*(P[i].y-Pk[g+4].y);
        Pk[g+4].y+=delta;
    }
  break;
  case 3://Nk-1
    switch (Cociente)
    {
      case 0://
        delta = a*(P[i].x-Pk[g-1].x);
        Pk[g-1].x+=delta;
        delta = a*(P[i].x-Pk[g+4].x);
        Pk[g+4].x+=delta;
        delta = a*(P[i].y-Pk[g-1].y);
        Pk[g-1].y+=delta;
        delta = a*(P[i].y-Pk[g+4].y);
        Pk[g+4].y+=delta;
        break;
      case 3://
        delta = a*(P[i].x-Pk[g-1].x);
        Pk[g-1].x+=delta;
        delta = a*(P[i].x-Pk[g-4].x);
        Pk[g-4].x+=delta;
        delta = a*(P[i].y-Pk[g-1].y);
        Pk[g-1].y+=delta;
        delta = a*(P[i].y-Pk[g-4].y);
        Pk[g-4].y+=delta;
        break;
      default:
        delta = a*(P[i].x-Pk[g-4].x);
        Pk[g-4].x+=delta;
        delta = a*(P[i].x-Pk[g-1].x);
```

```
        Pk[g-1].x+=delta;
        delta = a*(P[i].x-Pk[g+4].x);
        Pk[g+4].x+=delta;
        delta = a*(P[i].y-Pk[g-4].y);
        Pk[g-4].y+=delta;
        delta = a*(P[i].y-Pk[g-1].y);
        Pk[g-1].y+=delta;
        delta = a*(P[i].y-Pk[g+4].y);
        Pk[g+4].y+=delta;
    }
  break;
  default:
    switch (Cociente)
    {
      case 0://
        delta = a*(P[i].x-Pk[g-1].x);
        Pk[g-1].x+=delta;
        delta = a*(P[i].x-Pk[g+1].x);
        Pk[g+1].x+=delta;
        delta = a*(P[i].x-Pk[g+4].x);
        Pk[g+4].x+=delta;

        delta = a*(P[i].y-Pk[g-1].y);
        Pk[g-1].y+=delta;
        delta = a*(P[i].y-Pk[g+1].y);
        Pk[g+1].y+=delta;
        delta = a*(P[i].y-Pk[g+4].y);
        Pk[g+4].y+=delta;
        break;
      case 3://
        delta = a*(P[i].x-Pk[g-1].x);
        Pk[g-1].x+=delta;
        delta = a*(P[i].x-Pk[g+1].x);
        Pk[g+1].x+=delta;
        delta = a*(P[i].x-Pk[g-4].x);
        Pk[g-4].x+=delta;
        delta = a*(P[i].y-Pk[g-1].y);
        Pk[g-1].y+=delta;
        delta = a*(P[i].y-Pk[g+1].y);
        Pk[g+1].y+=delta;
        delta = a*(P[i].y-Pk[g-4].y);
        Pk[g-4].y+=delta;
        break;
      default:
        delta = a*(P[i].x-Pk[g-4].x);
        Pk[g-4].x+=delta;
        delta = a*(P[i].x-Pk[g+1].x);
        Pk[g+1].x+=delta;
        delta = a*(P[i].x-Pk[g-1].x);
        Pk[g-1].x+=delta;
        delta = a*(P[i].x-Pk[g+4].x);
        Pk[g+4].x+=delta;
        delta = a*(P[i].y-Pk[g-4].y);
        Pk[g-4].y+=delta;
        delta = a*(P[i].y-Pk[g+1].y);
        Pk[g+1].y+=delta;
        delta = a*(P[i].y-Pk[g-1].y);
        Pk[g-1].y+=delta;
        delta = a*(P[i].y-Pk[g+4].y);
        Pk[g+4].y+=delta;
    }
```

```
                    }
                    a=aux;
                }
            }
        }
        TColor color =clRed;
        if (AlmacenaPatron)color=clBlue;
        for (int x= Xmenor+1;x<Xmayor;x++)
        {
          OutData.SetPixel(x,Ymenor,color);
          OutData.SetPixel(x,Ymayor,color);
        }
        for (int y= Ymenor+1;y<Ymayor;y++)
        {
          OutData.SetPixel(Xmenor,y,color);
          OutData.SetPixel(Xmayor,y,color);
        }
        color = clBlack;
        for (int k=0; k<16;k++)
          OutData.SetPixel(int(Pk[k].x),int(Pk[k].y),color);
        if (AlmacenaPatron)
        {
          AlmacenaPatron=false;
          ofstream sal;
          sal.open("patrones",ios::app);
          if (sal)
          {
            float Ym=0;
            float Xm=0;
            for (int k=0;k<16;k++)
            {
              Xm += Pk[k].x;
              Ym += Pk[k].y;
            }
            Xm=Xm/16.0;
            Ym=Ym/16.0;
            for (int k=0;k<16;k++)
            {
              Pk[k].x-=Xm;
              Pk[k].y-=Ym;
            }
            for (int i =0 ;i<16;i++)
              sal << Pk[i].x << " " << Pk[i].y << " ";
            sal << Tipo << "\n";
            sal.close();
          }
          else
          {
            ShowMessage("No se pudo abrir el archivo");
          }
        }
      }
    }
  }
}
//---------------------------------------------------------------------
void __fastcall TForm1::VLGenericFilter3ProcessData(TObject *Sender,
              TVLCVideoBuffer InBuffer, TVLCVideoBuffer &OutBuffer,
```

```
                bool &SendOutputData)
{
  if (Region.Obejtos > 0)
  {
    //se ordenan los vehiculos segun el centro de masa
   EVehiculos VehAux;
   for (int i =1;i< Region.Obejtos+1;i++)
     for (int j =2;j< Region.Obejtos+1;j++)
       if (Vehiculos[i].X > Vehiculos[j].X)
       {
         VehAux = Vehiculos[j];
         Vehiculos[j]= Vehiculos[i];
         Vehiculos[i]=VehAux;
       }
   for (int i =1;i< Region.Obejtos+1;i++)
     if ((Vehiculos[i].Tipo >0)&&(Region.Xobjeto+5 <
          Vehiculos[i].X))
     {
       Region.Xobjeto = Vehiculos[i].X;
       if (Vehiculos[i].Tipo ==1) Region.Tipo1++;
       if (Vehiculos[i].Tipo ==2) Region.Tipo2++;
     }
  }
  else Region.Xobjeto = Region.Xmenor;
  tipo1->Caption = Region.Tipo1;
  tipo2->Caption = Region.Tipo2;
}
//---------------------------------------------------------------------
void __fastcall TForm1::coordenada(TObject *Sender, TMouseButton
                              Button,TShiftState Shift, int X, int Y)
{
  Region.Xmayor = X;
  Region.Ymayor = Y;
}
//---------------------------------------------------------------------
void __fastcall TForm1::coordenadamenor(TObject *Sender,
               TMouseButton Button, TShiftState Shift, int X, int Y)
{
  Region.Xmenor = X;
  Region.Ymenor = Y;
}
//---------------------------------------------------------------------
void __fastcall TForm1::VLGenericFilter7ProcessData(TObject *Sender,
               TVLCVideoBuffer InBuffer, TVLCVideoBuffer &OutBuffer,
               bool &SendOutputData)
{
  TVLCVideoDataAccess OutData = OutBuffer.Data();
  int delta = int(Region.Xmayor - Region.Xmenor)/4;
  int xd=Region.Xmenor -delta;
  xd = xd > 0 ? xd : 0;
  int xu = Region.Xmayor + delta;
  xu = xu>w? w : xu;
  delta = int(Region.Ymayor - Region.Ymenor)/2;
  int yd=Region.Ymenor -delta;
  yd = yd > 0 ? yd : 0;
  int yu = Region.Ymayor + delta;
  yu = yu>h? h : yu;
  for ( int y = 0; y < h ; y ++ )
```

```
    for ( int x = 0; x < w; x ++ )
      if (!(xu >= x && xd <= x && yu >= y && yd <= y))
        OutData.SetPixel(x,y,clBlack);

}
//---------------------------------------------------------------------
void __fastcall TForm1::AbrirVideoClick(TObject *Sender)
{
  Player->Paused =true;
  Open1->Execute();
  Player->FileName = Open1->FileName;
}
//---------------------------------------------------------------------
void __fastcall TForm1::PausaClick(TObject *Sender)
{
  if (Player->Paused==false)
  {
    Player->Paused =true;
    ObjMos=1;
    detener=false;
    Pausa->Caption = "Reanudar";
  }
  else
  {
    Player->Paused =false;
    Pausa->Caption = "Pausa";
  }
}
//---------------------------------------------------------------------
void __fastcall TForm1::ZonaClick(TObject *Sender)
{
  U1 = U1Edit->Text.ToInt();
  U3 = U3Edit->Text.ToInt();
  Uest = UestEdit->Text.ToInt();
  Region.Xobjeto = Region.Xmenor;
  Region.Obejtos=0;
  Region.Tipo1=0;
  Region.Tipo2=0;
}
//---------------------------------------------------------------------
void __fastcall TForm1::IniciarClick(TObject *Sender)
{
  Pausa->Enabled =true;
  Iniciar->Enabled = false;
  Frame =0;
  Player->CurrentFrame = 5;//IniFrame->Text.ToInt();
  Player->Paused =false;
}
//---------------------------------------------------------------------
void __fastcall TForm1::SpeedButton3Click(TObject *Sender)
{
  Close();
}
```