Proyecto Fin de Máster
Máster en Ingeniería Electrónica, Robótica y Automática.

Detección e identificación de objetos en movimiento en secuencias de vídeo.

Autor: Joaquín Barrachina Jordá
Tutor: Manuel Ruiz Arahal

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Proyecto Fin de Máster
Máster en Ingeniería Electrónica, Robótica y Automática

Detección e identificación de objetos en movimiento en secuencias de vídeo.

Autor:
Joaquín Barrachina Jordá

Tutor:
Manuel Ruiz Arahal
Profesor titular

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2021

Proyecto Fin de Máster: Detección e identificación de objetos en movimiento en secuencias de vídeo.

Autor: Joaquín Barrachina Jordá

Tutor: Manuel Ruiz Arahal

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2021

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

A mi familia por todo el apoyo y confianza depositada en mí, a mis amigos y compañeros por hacerlo todo más fácil, a mi tutor D. Manuel Ruiz Arahal por toda su ayuda y atención en el desarrollo de este proyecto y a todo el personal docente que a pesar de los tiempos de dificultad continuaron con su labor para que pudiésemos finalizar nuestros estudios en las mejores condiciones posibles.

Joaquín Barrachina Jordá

Graduado en Ingeniería Electrónica, Robótica y Mecatrónica

Sevilla, 2021

Resumen

En la actualidad se pueden encontrar gran multitud de aplicaciones, librerías y frameworks dedicados a la percepción automática que permiten el desarrollo de softwares avanzados sin la necesidad de conocer los fundamentos a partir de los que se obtienen los resultados.

En el presente proyecto se pretende realizar la construcción de una aplicación desde cero haciendo uso únicamente del software Matlab minimizando en la medida de lo posible el uso de funciones especializadas.

La aplicación construida recibirá una secuencia de vídeo donde deberá diferenciar, identificar y encerrar en una frontera a los diferentes objetos móviles que vayan apareciendo.

Abstract

At present, we can find a large range of applications, libraries and frameworks focused on automatic perception which allows to develop advanced software without the need of knowing anything about their basis.

In this project is intended to build an application from scratch only making use of the Matlab software minimizing as much as possible the use of specialised functions.

The built application will receive a video sequence where it will have to differentiate, identify, and enclose in a border the different mobile objects located in it.

Índice

Agradecimientos **i**

Resumen **iii**

Abstract **v**

Índice **vi**

Índice de Figuras **ix**

Notación **xiii**

1 Introducción **1**

2 Detección de movimiento entre fotogramas **4**

3 Tratamiento de la imagen **12**
3.1. Eliminación de píxeles por erosión *13*
3.2. Dilatación de píxeles *16*

4 Separación de objetos **21**

5 Obtención de fronteras **26**
5.1. Obtención de fronteras difusas *26*
5.2. Procesamiento de las fronteras *28*

6 Eliminación objetos interiores **32**

7 Identificación de objetos **37**
7.1. Extracción puntos característicos *37*
7.2. Emparejamiento de puntos singulares entre fotogramas *39*

8 Corrección de errores **44**

9 Generación de trayectorias **49**

10 Conclusiones y observaciones finales **57**

11 Anexos **63**
11.1. Estructura del algoritmo *63*
11.2. Carga de vídeo y extracción de parámetros *64*
11.3. Diferencia de imágenes *65*
11.4. Erosión de píxeles *67*
11.5. Dilatación de objetos *73*
11.6. Diferenciación de objetos *76*
11.7. Obtención fronteras difusas *79*
11.8. Procesado de fronteras difusas *81*
11.9. Eliminación de objetos interiores *82*

11.10. Obtención de puntos singulares *84*
11.11. Emparejamiento de puntos entre fotogramas *87*
11.12. Corrección de identificadores *90*
11.13. Construcción fotogramas resultado *91*
11.14. Generación de centros *94*
11.15. Obtención de las trayectorias *95*
11.16. Comparativa trayectorias obtenidas por algoritmo frente a ojo humano *101*

Referencias **109**

Glosario **111**

ÍNDICE DE FIGURAS

Figura 2-1. Muestra vídeo 'rhinos.avi' 5
Figura 2-2. Imagen en diferencia de dos fotogramas consecutivos 5
Figura 2-3. Imagen muestra del movimiento detectado 5
Figura 2-4. Ruido debido a cambio en la iluminación 6
Figura 2-5. Ejemplo imagen en diferencia 7
Figura 2-6. Ejemplo imagen en diferencia binaria 7
Figura 2-7. Imagen binaria sin umbral 8
Figura 2-8 Imagen binaria con umbral 20 8
Figura 2-9. Fotograma original 9
Figura 2-10. Imagen binaria, diferencia de imágenes 9
Figura 2-11. Imagen binaria, diferencia de gradientes de la imagen 9
Figura 3-1. Imagen en diferencia con píxeles residuales 12
Figura 3-2. Identificación de objetos con píxeles residuales 13
Figura 3-3. Limpieza de píxeles, vecindad 3x3 14
Figura 3-4. Limpieza de píxeles, vecindad 5x5 14
Figura 3-5. Imagen sin erosión aplicada 15
Figura 3-6. Imagen con erosión aplicada 15
Figura 3-7. Imagen con erosión aplicada, plantilla 22x22 16
Figura 3-8. Objeto segmentado, imagen en diferencia 16
Figura 3-9. Resultados objeto segmentado 17
Figura 3-10. Imagen en diferencia con acreción aplicada 18
Figura 3-11. Imagen resultado con acreción aplicada 18
Figura 3-12. Acreción de dimensión 16 sobre imagen en diferencia 19
Figura 3-13. Imagen resultado con acreción de dimensión 16. 19
Figura 4-1. Primer píxel localizado 22
Figura 4-2. Actualización de píxel localizado y vecindad 22
Figura 4-3. Actualización vecindad de píxel almacenado en registro 22
Figura 4-4. Localización de nuevo objeto 23
Figura 4-5. Caso de fusión de objetos durante inundado 23
Figura 4-6. Resultado de fusión de objetos 24
Figura 5-1. Localización de píxel perteneciente a objeto 27
Figura 5-2. Obtención de frontera que rodea al píxel localizado 27
Figura 5-3. Frontera obtenida tras procesamiento de todos los píxeles 27

Figura 5-4. Trazado de fronteras difusas sobre fotograma 28
Figura 5-5. Valores límite de la frontera difusa 29
Figura 5-6. Resultado de empleo de los valores límite 29
Figura 5-7. Fronteras procesadas sobre fotograma de vídeo 30
Figura 6-1. Objeto interior detectado 32
Figura 6-2. Objeto fragmentado a pesar de tratamiento 33
Figura 6-3. Objeto interior cumpliendo las 4 condiciones dadas 34
Figura 6-4. Objeto parcialmente interior cumpliendo 3 de las 4 condiciones dadas 34
Figura 6-5. Resultado de aplicar la eliminación de objetos interiores 35
Figura 7-1. Puntos característicos extraídos en fotograma 39
Figura 7-2. Puntos característicos en primer fotograma de pareja consecutiva 40
Figura 7-3. Puntos característicos en segundo fotograma de pareja consecutiva 40
Figura 7-4. Identificación de objetos 41
Figura 7-5. Identificación de objetos, aparición de nuevo objeto 42
Figura 8-1. Dos objetos fusionados como uno reciben identificador 44
Figura 8-2. Objetos dejan de estar fusionados, pero comparten identificador 45
Figura 8-3. Objetos fusionados como uno reciben identificador 45
Figura 8-4. Objetos dejan de estar fusionados y reciben identificadores distintos 46
Figura 8-5. Recta a partir de la cual se comenzará la identificación 46
Figura 8-6. Identificación de primer objeto que atraviesa la línea 47
Figura 8-7. Identificación de múltiples objetos al atravesar la línea 47
Figura 9-1. Trazado de centro del objeto 49
Figura 9-2. Unión de puntos simple, ejemplo 1 50
Figura 9-3. Unión de puntos simple, ejemplo 2 50
Figura 9-4. Unión suavizada 51
Figura 9-5. Fotograma resultado con trayectorias incorporadas, ejemplo 1 51
Figura 9-6. Fototgrama resultado con trayectorias incorporadas, ejemplo 2 52
Figura 9-7. Graficación trayectorias de todos los vehículos 52
Figura 9-8. Comparativa trayectoria obtenida por algoritmo y trayectoria obtenida por humano 53
Figura 9-9. Error entre las trayectorias obtenidas por algoritmo y por humano, ejemplo 1 53
Figura 9-10. Comparativa trayectoria obtenida por algoritmo y trayectoria obtenida por humano 54
Figura 9-11. Error entre las trayectorias obtenidas por algoritmo y por humano, ejemplo 2 54
Figura 9-12. Muestras incompletas en la trayectoria obtenida por el experto humano 55
Figura 10-1. Píxeles de ruido detectados como objetos 58
Figura 10-2. Objeto en movimiento segmentado en varios 58
Figura 10-3. Resultados de vídeo paseo, ejemplo 1 59
Figura 10-4. Resultados vídeo paseo, ejemplo 2 59
Figura 10-5. Resultados de vídeo paseo, ejemplo 3 60

Figura 11-1. Ventanas desbordan límites de la imagen 68
Figura 11-2. Ventanas adaptativas, no desbordan límites de la imagen 68
Figura 11-3. Conversión ventana tamaño fijo a ventana tamaño adaptativo 71
Figura 11-4. Fotograma previo a la dilatación de objetos 74
Figura 11-5. Fotograma tras la dilatación de objetos 74
Figura 11-6. Error al actualizar estados en matriz en vez de utilizar una nueva 75
Figura 11-7. Comparativa trayectorias vehículo 1 101
Figura 11-8. Error entre trayectorias vehículo 1 101
Figura 11-9. Comparativa trayectorias vehículo 2 101
Figura 11-10. Error entre trayectorias vehículo 2 102
Figura 11-11. Comparativa trayectorias vehículo 3 102
Figura 11-12. Error entre trayectorias vehículo 3 102
Figura 11-13. Comparativa trayectorias vehículo 4 103
Figura 11-14. Error entre trayectorias vehículo 4 103
Figura 11-15. Comparativa trayectorias vehículo 5 103
Figura 11-16. Error entre trayectorias vehículo 5 104
Figura 11-17. Comparativa trayectorias vehículo 6 104
Figura 11-18. Error entre trayectorias vehículo 6 104
Figura 11-19. Comparativa trayectorias vehículo 7 105
Figura 11-20. Error entre trayectorias vehículo 7 105
Figura 11-21. Comparativa trayectorias vehículo 8 105
Figura 11-22. Error entre trayectorias vehículo 8 106
Figura 11-23. Comparativa trayectorias vehículo 9 106
Figura 11-24. Error entre trayectorias vehículo 9 106
Figura 11-25. Comparativa trayectorias vehículo 10 107
Figura 11-26. Error entre trayectorias vehículo 10 107

Notación

A^*	Conjugado
c.t.p.	En casi todos los puntos
c.q.d.	Como queríamos demostrar
■	Como queríamos demostrar
e.o.c.	En cualquier otro caso
e	número e
IRe	Parte real
IIm	Parte imaginaria
sen	Función seno
tg	Función tangente
arctg	Función arco tangente
sen	Función seno
$\sin^x y$	Función seno de x elevado a y
$\cos^x y$	Función coseno de x elevado a y
Sa	Función sampling
sgn	Función signo
rect	Función rectángulo
Sinc	Función sinc
∂y ∂x	Derivada parcial de y respecto
$x_\circ$	Notación de grado, x grados.
Pr(A)	Probabilidad del suceso A
SNR	Signal-to-noise ratio
MSE	Minimum square error
:	Tal que
<	Menor o igual
>	Mayor o igual
\	Backslash
⇔	Si y sólo si

1 INTRODUCCIÓN

La ciencia es el padre del conocimiento, pero las opiniones son las que engendran la ignorancia.

- Hipócrates -

En la actualidad se pueden encontrar una gran diversidad de campos donde se precisan aplicaciones para la detección de movimiento. Con el fin de lograr esta tarea se puede recurrir a múltiples métodos entre los que se incluyen:

- Detección infrarroja. Sensores perciben la presencia de cuerpos en movimiento por medio de la energía calorífica que emana de estos.
- Detección magnética. Sensores perciben movimiento mediante la separación de componentes imantados.
- Detección por vibración. Se detetcta el movimiento por medio de sensores que perciben las oscilaciones de los cuerpos.
- Detección por energía de radio frecuencia. Sensor detecta movimiento por medio de la alteración de la frecuencia en una señal que él mismo emite.
- Detección óptica. Se lleva a cabo la detección de movimiento haciendo uso de sistemas de cámaras o directamente secuencias de vídeo para percibir cuerpos móviles.

En el presente proyecto se abordará este último método, la detección óptica, ya que es una tecnología cuya funcionalidad se encuentra en constante crecimiento y está siendo cada vez más empleada en aplicaciones como:

- Aplicaciones de seguridad. Cámaras de vídeo se emplean para detección de movimiento de animales o posibles intrusos.
- Aplicaciones militares. En la detección de objetivos desde aeronaves o en el seguimiento de blancos.
- Aplicaciones en teléfonos móviles. Detección del usuario para poder acceder al teléfono móvil.

Hoy en día, los algoritmos que se esconden tras estas aplicaciones se encuentran ampliamente desarrollados sobre una gran variedad de librerías con funciones especializadas como podrían ser la librería openCV basada en Python o la librería de TensorFlow YOLO Object Detection. En estas librerías se incluyen una alta gama de funciones que permiten la construcción de aplicaciones de una manera sencilla sin que sea necesario entender cómo funcionan las bases del procesamiento de imagen.

Es por eso por lo que la motivación de este trabajo se basará en la construcción de un algoritmo de detección de movimiento en una secuencia de vídeo desde cero haciendo uso del software Matlab, tratando de evitar en la medida de lo posible el empleo de funciones especializadas para el procesamiento de vídeo.

Los pasos seguidos para la construcción de este algoritmo serán los siguientes:

1. Detección de movimiento entre fotogramas consecutivos.
2. Tratamiento de la imagen para optimización de los resultados.
3. Separación de objetos en movimiento detectados.
4. Identificación de objetos en movimiento detectados.
5. Seguimiento de objetos detectados a lo largo de la secuencia de vídeo.
6. Modificación sobre los fotogramas para visualización de resultados.

El desarrollo de esta aplicación se elaboró en torno al vídeo 'traffic.mj2' perteneciente a las librerías de Matlab, la razón por la que se escogió este vídeo se desarrolla en el siguiente punto del proyecto. En este vídeo se observa una grabación realizada por una cámara de tráfico desde una posición fija sobre un tramo de una carretera por la que circulan coches en un solo sentido.

La carga de este vídeo en la aplicación y la estructura principal del algoritmo se pueden consultar en los anexos 1 y 2 de este mismo documento.

2 DETECCIÓN DE MOVIMIENTO ENTRE FOTOGRAMAS

El aspecto más triste de la vida en este preciso momento
es que la ciencia reúne el conocimiento más rápido de
lo que la sociedad reúne la sabiduría.

- Isaac Asimov -

El punto de partida para el desarrollo de la aplicación de detección y seguimiento de objetos en una secuencia de vídeo sería la detección de movimiento entre dos fotogramas consecutivos. El procedimiento empleado para llevar a cabo esta tarea ha sido el método de diferencia entre imágenes [1]. Este es un método común para la detección de movimiento que se basa en calcular la diferencia de intensidades entre los dos fotogramas consecutivos píxel a píxel de forma que los píxeles resultado que tengan un valor próximo a 0 se asociarán a que no se ha producido movimiento sobre estos y viceversa, cuando el resultado se aleje de cero se deberá a que ha cambiado la intensidad de un fotograma a otro, por lo que ha habido algún tipo de movimiento.

Aunque la diferencia de imágenes es un método sencillo este se encuentra sometido a limitaciones y restricciones que se deben tener en cuenta. Las principales son las siguientes:

1. Empleo de cámara fija.

Los vídeos que se emplean para ser sometidos a este método de detección tienen que haber sido grabados por medio de una cámara en una posición estática. De esta forma se detectan como objetos en movimiento cualquier elemento que se mueva delante de la cámara, de no ser así y encontrarse la cámara el movimiento lo que sucedería sería que el método de diferencia entre imágenes identificaría como objetos móviles cualquier cosa que apareciese en el vídeo, aunque esta estuviese inmóvil como veremos a continuación.

Esto se comprobó en una fase más avanzada del proyecto donde ya se podía realizar la detección de movimiento, se empleó como vídeo para realizar la detección uno de los pertenecientes al toolbox de Matlab, 'rhinos.avi'. En este vídeo un cámara desde el interior de su vehículo sigue el paseo de dos rinocerontes. El resultado producido fue que además de no terminar de capturar bien el movimiento de los rinocerontes, solo se detectaba el movimiento de sus piernas, también capturaba muchos elementos inmóviles como las copas de los árboles del paisaje o el propio salpicadero del vehículo que en realidad no se estaban moviendo.

Figura 2-1. Muestra vídeo ‘rhinos.avi’.

Figura 2-2. Imagen en diferencia de dos fotogramas consecutivos.

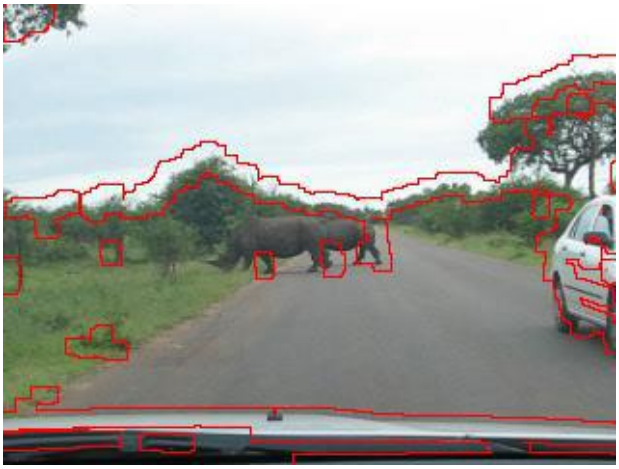

Figure 2-3. Imagen muestra del movimiento detectado

2. Método sensible a cambios de iluminación.

Otro punto negativo de este método sería su alta sensibilidad a los cambios de iluminación, como resultado de aplicar la diferencia de imágenes a dos fotogramas con distinto brillo se obtiene una imagen en diferencia con muchos píxeles que se interpretan como ruido. Esto fuerza la inclusión de funciones de preprocesamiento que nos permitan su limpieza. A continuación, figura 2-4, se muestra un ejemplo de este efecto encontrado en el desarrollo de la aplicación sobre algunos de los fotogramas del vídeo 'traffic.mj2'.

Figura 2-4. Ruido debido a cambio en la iluminación.

Teniendo en cuenta estas consideraciones la aplicación del método es muy sencilla de llevar a cabo, los pasos serían los siguientes:

1. Extracción de dos fotogramas consecutivos del vídeo.
2. Conversión de los fotogramas RGB a escala de grises.
3. Aplicación de la diferencia entre los fotogramas en escala de grises.
4. Conversión de la imagen en diferencias a imagen binaria.

Este último punto sería el único en el que resulta interesante entrar en detalle ya que aquí encontramos un parámetro de diseño para nuestra aplicación, el umbral para convertir la imagen en diferencia en una imagen binaria.

Este umbral se emplea para determinar a partir de que valor de intensidad obtenido en la imagen en diferencia se considerará como píxel perteneciente a objeto en movimiento o no, de forma que los píxeles ahora pasarán exclusivamente a tener valores de cero y uno, cero en caso de que el píxel no sobrepase el umbral escogido y uno en caso de que sí se sobrepase este valor.

Figura 2-5. Ejemplo imagen en diferencia.

Figura 2-6. Ejemplo imagen en diferencia binaria.

Este valor umbral no se debe escoger a la ligera, ya que cuanto mayor se haga se eliminarán más píxeles asociados a ruido, pero también comenzarán a desaparecer píxeles asociados a objetos en movimiento. Se deberán entonces de realizar pruebas para escoger un umbral que filtre la mayor cantidad de ruido posible afectando lo menos posible a los objetos.

A continuación, se incluye un ejemplo comparativo entre no aplicar ningún valor umbral, figura 2-7, y aplicar un umbral de 20, figura 2-8.

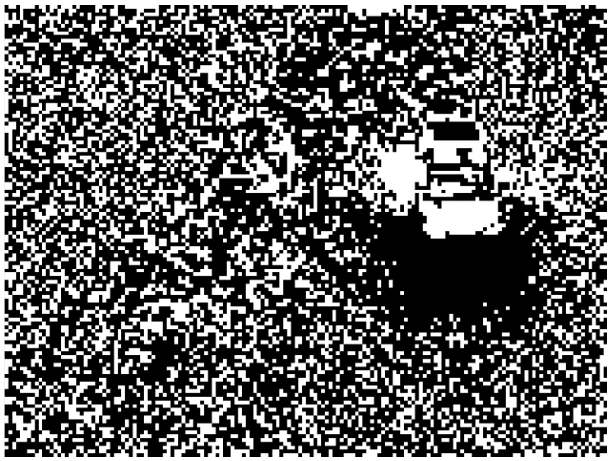

Figura 2-7. Imagen binaria sin umbral.

Figura 2-8. Imagen binaria con umbral 20.

En una primera instancia fue de esta manera como se construyó la función para el proyecto, pero más adelante conforme se fue avanzando en este los resultados no se terminaban de ver óptimos y en gran parte el problema surgía de que algunos objetos en esta función quedaban segmentados y no había forma de en funciones posteriores hacerlos ver como uno solo. Tras realizar varios experimentos se encontró una solución viable, esta consistía en aplicar la diferencia de imágenes, pero no sobre la imagen en sí, sino sobre su módulo de gradiente. Por tanto, se añadieron como pasos adicionales la obtención de los módulos de gradiente de los fotogramas mediante el método de Sobel y se aplicó la diferencia entre estos, por último, se aplicaría un nuevo umbral para la obtención de la imagen binaria. Con esto se obtuvieron cambios notorios como el observado en la figura 2-11.

Figura 2-9. Fotograma original.

Figura 2-10. Imagen binaria, diferencia de imágenes.

Figura 2-11. Imagen binaria, diferencia de gradientes de la imagen.

Se puede observar como el maletero de la figura 9 queda dividido en dos objetos separados a una cierta distancia en la figura 10, mientras que en la figura 11 aunque está dividido en pequeños fragmentos se contempla mejor la forma del objeto original. Además, estos pequeños objetos que conforman el total podrán fusionarse con funciones de procesamientos que se construirán más adelante.

El código Matlab correspondiente a esta función se encuentra explicado en el Anexo 3.

3 TRATAMIENTO DE LA IMAGEN

La ciencia aumenta nuestro poder en la medida en que reduce nuestra soberbia.

- Herbert Spencer -

Obtenida la imagen en diferencia el siguiente paso será la construcción de funciones de preprocesamiento que reduzcan, en la medida de lo posible, errores para la posterior detección de objetos. Algunos de estos errores son:

1. Detección de objetos en movimiento cuando en realidad no los hay.
2. Único objeto segmentado en varios.
3. Varios objetos fusionados en uno solo.

La causa del primer error, la obtención de falsos positivos en la secuencia de vídeo se debe, en la mayoría de las ocasiones, a píxeles residuales pertenecientes a ruido, como hemos visto con anterioridad, que deben ser eliminados para facilitar la separación de regiones.

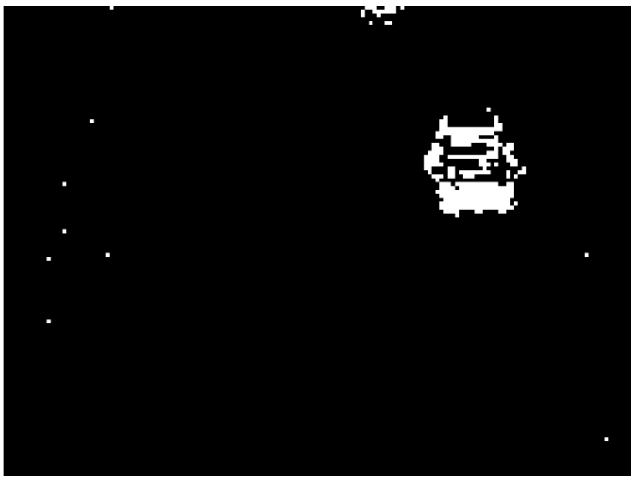

Figura 3-1. Imagen en diferencia con píxeles residuales.

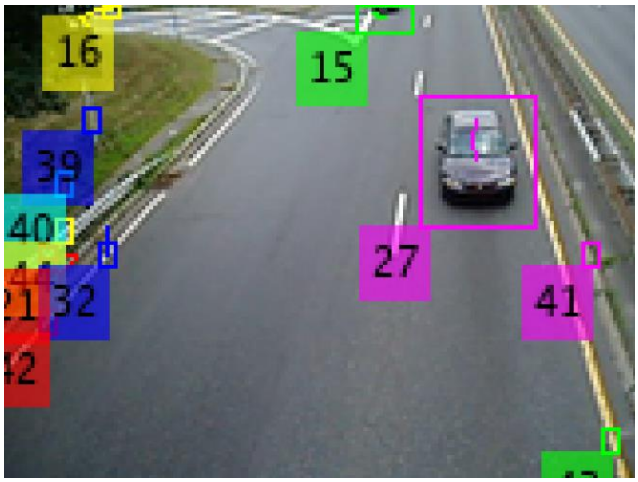

Figura 3-2. Identificación de objetos con píxeles residuales.

Se puede contemplar en estas imágenes como el no realizar la limpieza de los píxeles individuales, o de pequeñas agrupaciones de estos, puede perjudicar gravemente el resultado final de la aplicación.

Anteriormente se había realizado un primer barrido de este ruido por medio de una correcta elección del valor umbral que convertía la imagen en diferencia en imagen binaria, con este valor es posible realizar una calibración que permite obtener una imagen binaria limpia de este ruido directamente. Un problema que se encontró fue que para el valor umbral en el que se consigue la eliminación total del ruido también se perdía demasiada información de píxeles pertenecientes a objetos.

Será entonces cuando en vez de ajustar este parámetro umbral hasta estos valores extremos se optará por la aplicación de un método morfológico, la erosión, sobre la imagen binaria obtenida. Los métodos morfológicos en este caso particular mejorarán la separación de regiones haciendo uso de operaciones no lineales.

Se construye entonces una función encargada de aplicar este primer método mencionado para la eliminación de píxeles.

3.1 Eliminación de pixels por erosión.

La idea que persigue implementar esta función es la siguiente, se buscarán píxeles aislados o pequeñas agrupaciones de estos y una vez identificados como tal pasarán a tener de una valor 1 a un valor 0.

Se comienza entonces leyendo la imagen binaria obtenida en el primer apartado. Durante esta lectura cada vez que se encuentre un píxel con valor de uno y no de cero se realizará la siguiente comprobación: si en el perímetro de píxeles que rodean al que está siendo analizado tienen todos un valor de cero, se interpretará el píxel como uno aislado y se pondrá este también a cero. La representación gráfica de esta idea es la siguiente:

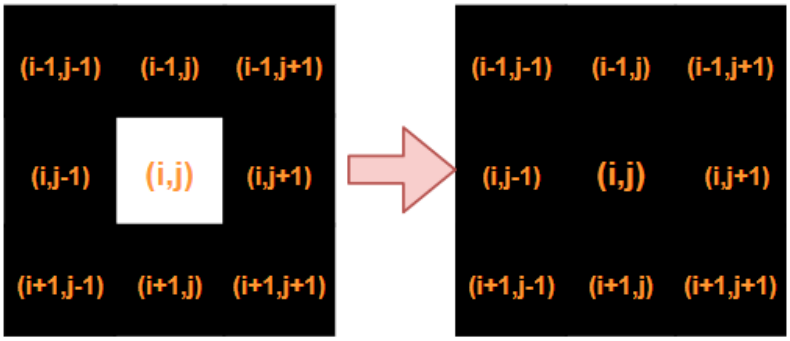

Figura 3-3. Limpieza de píxeles, vecindad 3x3.

Una vez se conseguida la limpieza de estos píxeles se extendió la funcionalidad de la función para poder también realizar la limpieza de motas de píxeles tan grandes como se desease. Pasaría entonces a ser otro parámetro de diseño la ventana de limpieza escogida, la idea es la misma que la mostrada en el caso anterior: una vez leído un píxel con valor a uno leemos ahora no su vecindad 3x3, sino que comprobaremos el perímetro de la ventana escogida, de forma que sí en este perímetro todos los píxeles tienen valor cero todos los píxeles del interior de este perímetro pasarán también a tener valor cero. A continuación, un ejemplo gráfico de escoger para la limpieza una ventana de lado 5.

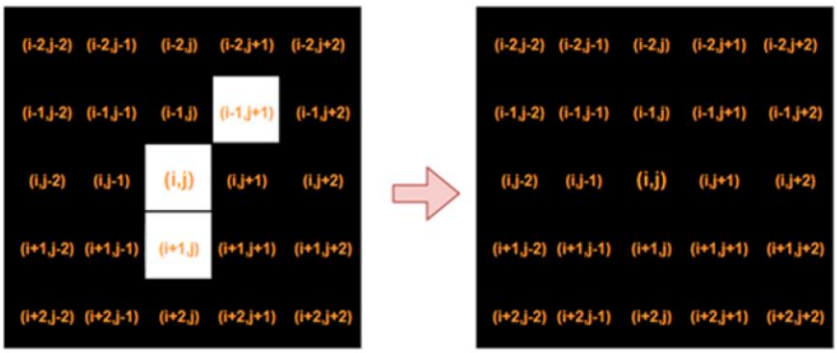

Figura 3-4. Limpieza de píxeles, vecindad 5x5.

En resumen, en este proceso hemos aplicado el método morfológico de erosión, pero de forma diferente a la comúnmente conocida, ecuación 3-1, en el método desarrollado en vez de aplicar una plantilla predefinida a cada píxel de la imagen aplicamos la mencionada ventana únicamente en los píxeles con valor distinto de 0.

$$Ip = Io \ominus p \tag{3–1}$$

En la ecuación de la erosión mostrada 'Ip' hace referencia a la imagen procesada tras la aplicación del método morfológico, 'Io' sería la imagen original y 'p' la plantilla empleada.

La implementación de esta función se ha preparado de tal forma que la ventana se pueda hacer tan grande como se deseé, pero se deberá tener en cuenta que cuanto mayor se haga mayores probabilidades habrán de que se eliminen píxeles pertenecientes a objetos u objetos en sí.

En el desarrollo de la aplicación se optó por la selección de una ventana máxima de limpieza de 5x5 de forma que se eliminasen tanto píxeles individuales como motas de hasta 3x3.

Figura 3-5. Imagen sin erosión aplicada.

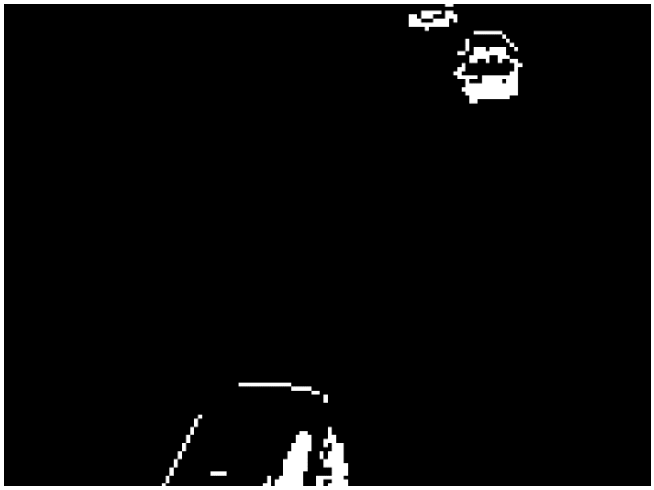

Figura 3-6. Imagen con erosión aplicada.

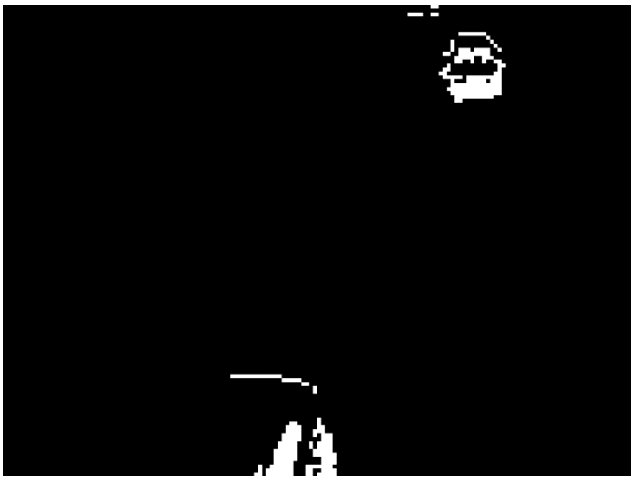

Figura 3-7. Imagen con erosión aplicada, plantilla 22x22.

El código Matlab correspondiente a esta función se encuentra explicado en el Anexo 4.

3.2 Dilatación de píxeles

Otro de los problemas que se detectó en los resultados conforme se avanzaba en el desarrollo del proyecto era la identificación de múltiples objetos como partes independientes de lo que realmente se correspondía con un solo objeto. Este error se producía debido a que el método de detección de objetos construido más adelante precisa que los píxeles pertenecientes a un mimo objeto estén pegados entre sí dentro de una vecindad 3x3, en caso contrario pasan a identificarse como objetos independientes, esto se observa en las figuras siguientes.

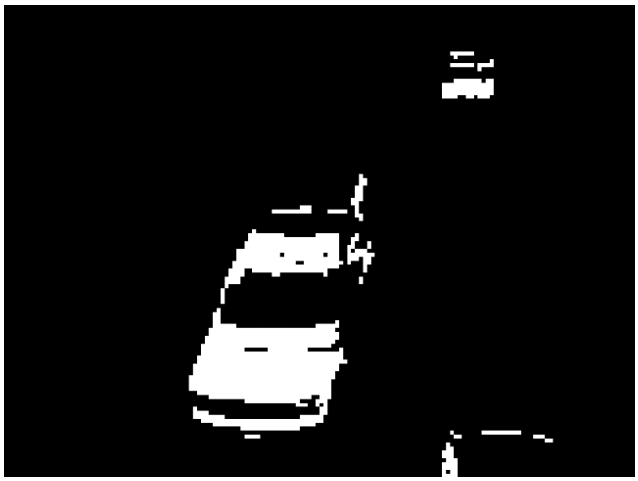

Figura 3-8. Objeto segmentado, imagen en diferencia.

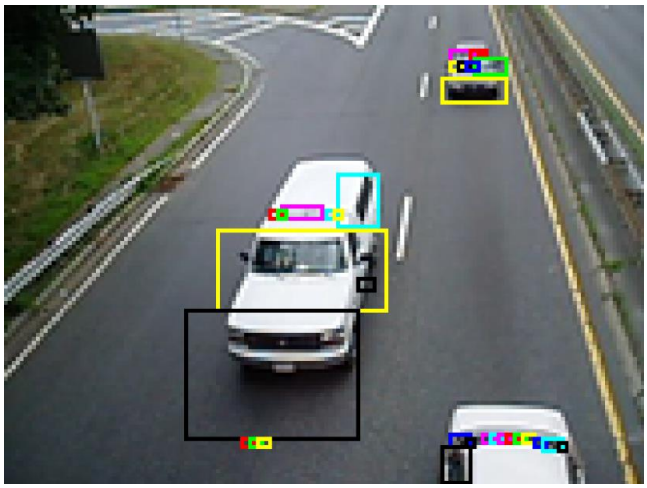

Figura 3-9. Resultados objeto segmentado.

Se puede ver como nuevamente la aplicación no funciona correctamente si no se atiende este error. La solución que se le dio a este fue, además de jugar con los parámetros de diseño vistos hasta ahora, engordar los píxeles que han llegado hasta este punto. Los píxeles que se observan con valor a uno en la imagen ya han pasado unos determinados umbrales y procesamientos por lo que se consideran parte de algún objeto en movimiento.

Para obtener este engordado, o dilatación, de píxeles se recurrirá a otro método morfológico comúnmente usado, la acreción.

Se construirá entonces una nueva función con la finalidad de aplicar este método morfológico sobre cada píxel en la imagen con un valor de 1. En esta función se escogerá cuanto se desean dilatar los píxeles por lo que cada uno de ellos pasará de tener, en función de la ventana escogida, una dimensión 1x1 a 3x3, 5x5…

Nuevamente, al igual que en el caso de la erosión, no se ha aplicado el método morfológico de forma convencional, ecuación 3-2, si no que volvemos a aplicar una ventana únicamente en píxeles con un valor distinto de 0 en lugar de una plantilla específica para cada píxel de la imagen.

$$Ip = Io \oplus p \tag{3–2}$$

Con esto se consigue que partes de un mismo objeto cercano se fusionen formando uno solo con lo que se resuelve el problema presentado.

Figura 3-10. Imagen en diferencia con acreción aplicada.

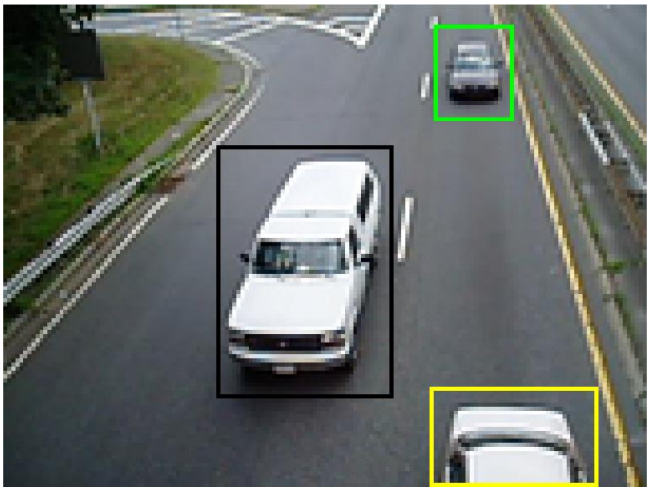

Figura 3-11. Imagen resultado con acreción aplicada.

El parámetro que regula el incremento de la dimensión del píxel será tratado como otro parámetro de diseño de la aplicación con el que se podrán realizar diversas pruebas. Es importante que este se escoja con precaución ya que de dejarlo en un valor pequeño se podrían seguir encontrando problemas de objetos fragmentados o, en el caso contrario, de escogerse muy grande, se podría contemplar un nuevo error, que al incrementar mucho la dimensión de los píxeles pertenecientes a objetos varios de ellos podrían fusionarse en uno solo.

Fiugra 3-12. Acreción de dimensión 16 sobre imagen en diferencia.

Figura 3-13. Imagen resultado con acreción de dimensión 16.

El valor escogido para el parámetro de engorde en la aplicación final fue de una unidad, aumentándose la dimensión de los píxeles de 1x1 a su vecindad 3x3.

El código Matlab correspondiente a esta función se encuentra explicado en el Anexo 5.

4 SEPARACIÓN DE OBJETOS

Cuanto mas original es un descubrimiento, más obvio parece después.

- Arthur Koestler -

Habiendo llegado a este punto de la aplicación se cuenta ya con un fotograma binario que ha sido procesado de manera que todos lo píxeles con valor de uno se consideran pertenecientes a un objeto y a cero los que no.

En este apartado se pasará a abordar el desafío de como identificar si los píxeles del fotograma pertenecen a un objeto en movimiento u otro para así conseguir realizar la separación de estos objetos y su recuento total.

La idea llevada a cabo es la siguiente, se comenzarán a leer los píxeles de la imagen hasta que se encuentre un píxel con valor igual a uno. El encontrar un píxel con este valor indica la presencia de un nuevo objeto que aún no ha sido procesado.

Una vez localizado uno de estos píxeles se le asociará un nuevo valor, este valor se escoge por medio de una variable contador que se va incrementando cada vez que se localiza un nuevo objeto en el fotograma, en el caso de la primera vez que un objeto es encontrado este contador ha de empezar en 2 ya que si se le diese valor de 1 el algoritmo lo volvería a interpretar en futuras iteraciones como píxel no procesado.

Inicializado el píxel con este nuevo valor comenzará a desarrollarse un proceso de inundado para buscar píxeles pertenecientes al mismo objeto. Se comienza entonces a observar la vecindad 3x3 del píxel para ver si en esta hay nuevamente píxeles con un valor de 1, no procesados. En caso afirmativo los valores de estos píxeles serían actualizados al mismo valor que el píxel original y las coordenadas de estos serían almacenadas en unos registros para su posterior análisis. Finalizada la iteración actual no volveríamos directamente a la estructura principal de lectura del fotograma, en su lugar accedemos a comprobar si en los registros se han almacenado nuevos píxeles y en caso afirmativo se procedería a repetir el proceso anterior para cada uno de estos.

Se entraría entonces en un bucle que irá leyendo los registros y procesando los píxeles en su interior, en cada iteración nuevos píxeles serán almacenados en estos registros de forma que este bucle seguirá funcionando hasta que se detecte que en los registros ya no se ha producido ninguna nueva entrada y que todos los píxeles en su interior ya han sido analizados, destacar que al cambiar el valor de los píxeles a la vez que se guardan en los registros estos no volverán a almacenarse de forma duplicada en estos al tener en futuros encuentros un valor distinto de 1.

Finalizada la lectura de todos los píxeles en los registros se abandonaría este bucle para volver a la estructura principal de lectura de píxeles de la imagen. Ahora el algoritmo pasará por alto todos los píxeles procesados como pertenecientes al objeto anterior, recordemos que hemos actualizado su valor de 1 a 2 por lo que se considerarán como ya analizados. Entonces este bucle seguiría hasta la aparición de un nuevo píxel a 1 que marcaría la aparición de un nuevo objeto con lo que se comenzaría de nuevo el proceso explicado con anterioridad, pero estos píxeles tendrían un valor distinto a los del objeto anterior lo que garantizaría su diferenciación.

Todo este proceso de inundado se encuentra representado gráficamente en las siguientes figuras.

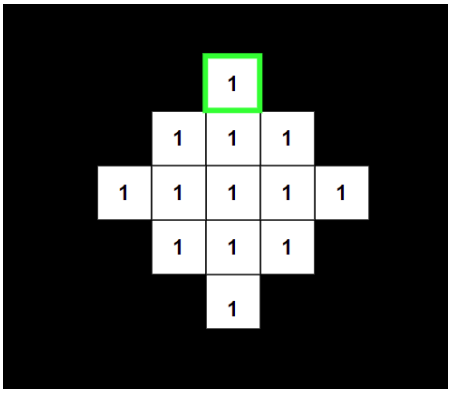

Figura 4-1. Primer píxel de objeto localizado.

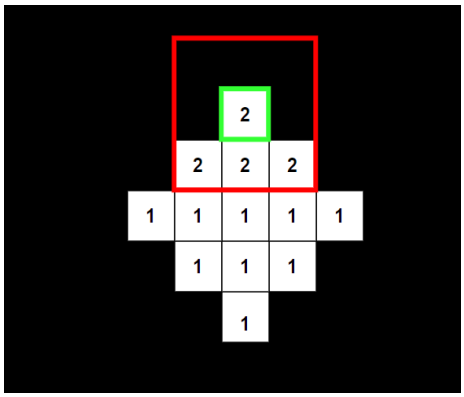

Figura 4-2. Actualización de píxel localizado y vecindad.

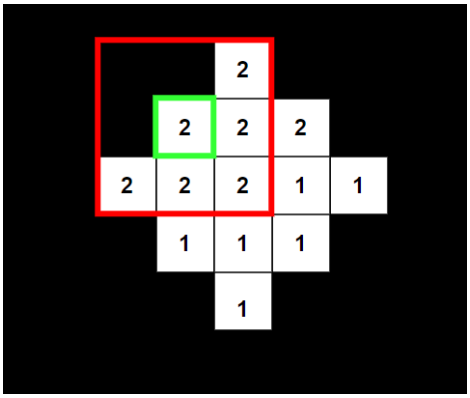

Figura 4-3. Actualización vecindad de píxel almacenado en registro.

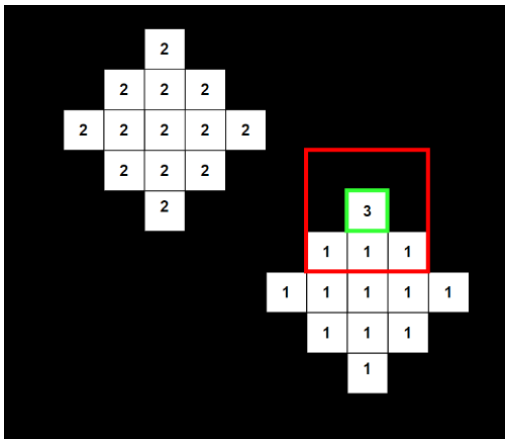

Figura 4-4. Localización de nuevo objeto.

Como se puede observar para el correcto funcionamiento de esta función es imprescindible que los píxeles que conformen un objeto estén unidos entre sí por al menos un píxel en la vecindad 3x3 para garantizar la continuidad del proceso de inundado y que el objeto no aparezca segmentado como varios. También es fundamental que no estén en contacto dentro de esta vecindad con píxeles de otros objetos para evitar que este también sea inundado y considerado como uno solo a los ojos del algoritmo. De aquí la importancia que hemos dado al correcto diseño de las funciones de preprocesado en los apartados anteriores.

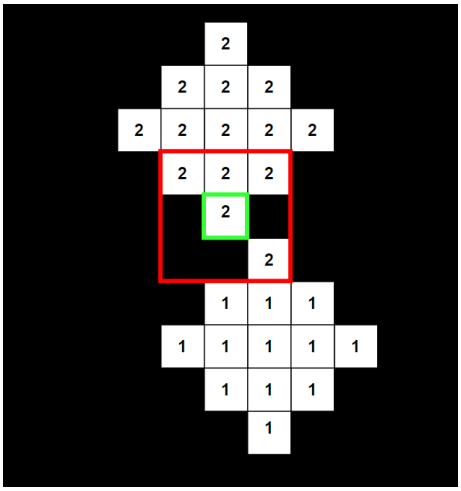

Figura 4-5. Caso de fusión de objetos durante inundado.

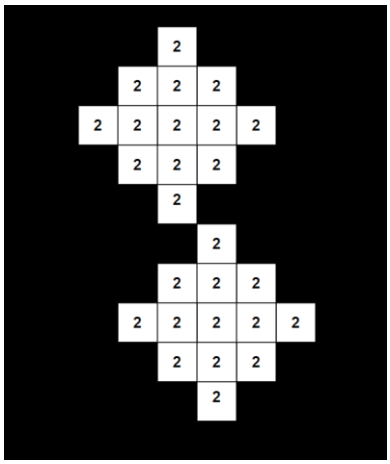

Figura 4-6. Resultado de fusión de objetos.

El código Matlab correspondiente a este apartado se desarrolla en detalle en el anexo 6.

5 OBTENCIÓN DE FRONTERAS

Los que dicen que es imposible no deberían molestar a los que lo están haciendo.

- Albert Einstein -

Hasta ahora en el proceso del algoritmo ya hemos conseguido obtener un fotograma procesado en el que se han identificado objetos en movimiento y se han diferenciado estos entre sí. Pero aún no se ha aplicado ninguna modificación que haga visible estos procesamientos sobre los fotogramas de la secuencia de vídeo original, por tanto, se comenzarán a trazar fronteras que encierren a los objetos en movimiento que se han diferenciado en el fotograma. Este proceso consta de dos apartados:

1- Obtención de las fronteras difusas.
2- Procesamiento de las fronteras difusas para la obtención de fronteras rectangulares.

5.1 Obtención de fronteras difusas

El algoritmo comienza entonces a leer el fotograma que contiene a los objetos diferenciados, recordemos que cada objeto tiene sus píxeles a un determinado valor y que este valor es único para el objeto. Es en el momento en el que se encuentra un píxel con valor distinto de cero que se comienza con el trazado de la frontera difusa.

Localizado el píxel de un objeto se observará su vecindad 3x3, dentro de esta, todo píxel con valor de cero será considerado como un píxel libre que puede usarse como píxel frontera. Este píxel o píxeles encontrados como libres en la vecindad serán entonces almacenados en una estructura de datos para poder ser procesados más adelante, esta estructura contendrá las fronteras independientes de cada uno de los objetos en el fotograma ya que en esta se irán almacenando sus respectivas coordenadas.

La representación de este proceso se muestra en las siguientes figuras:

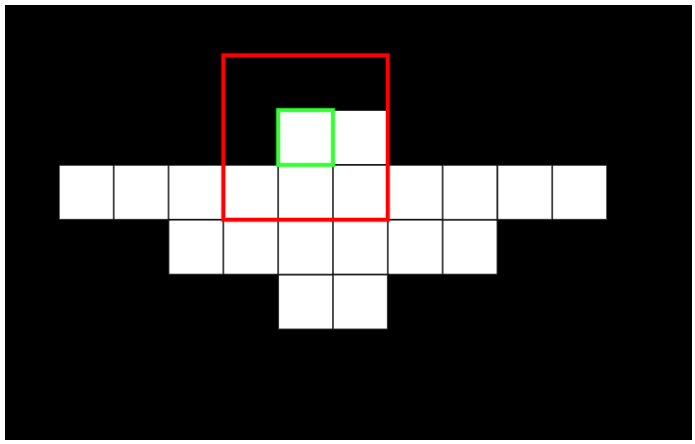

Figura 5-1. Localización de píxel perteneciente a objeto.

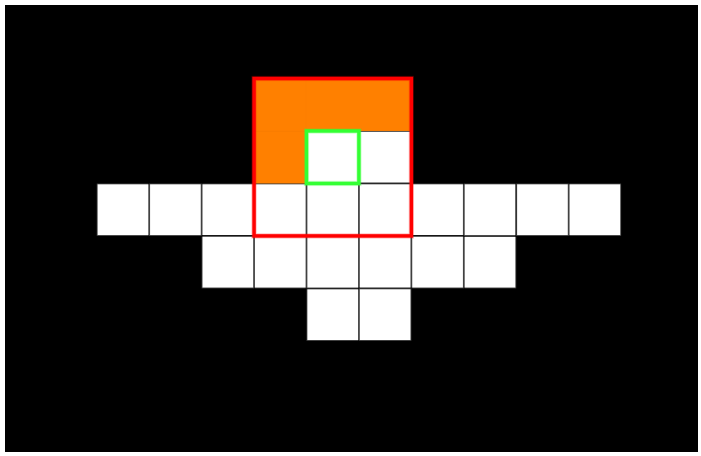

Figura 5-2. Obtención de frontera que rodea al píxel localizado.

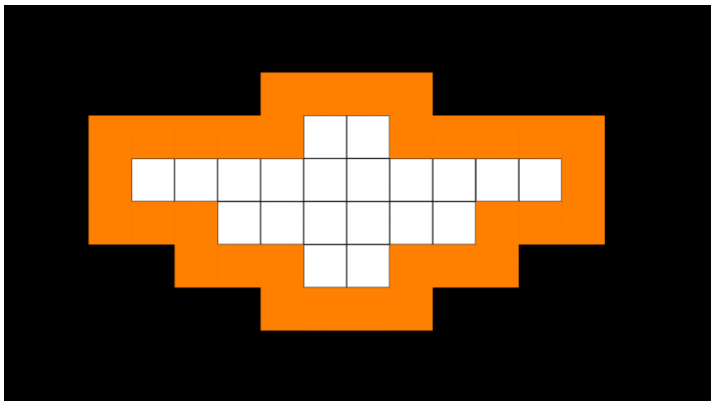

Figura 5-3. Frontera obtenida tras procesamiento de todos los píxeles.

Aunque estas fronteras no son las definitivas que se usarán en la aplicación para comprobar el correcto funcionamiento llegados a este punto se trazaron estas sobre los fotogramas del vídeo dando resultados como el observado en la figura 5-4.

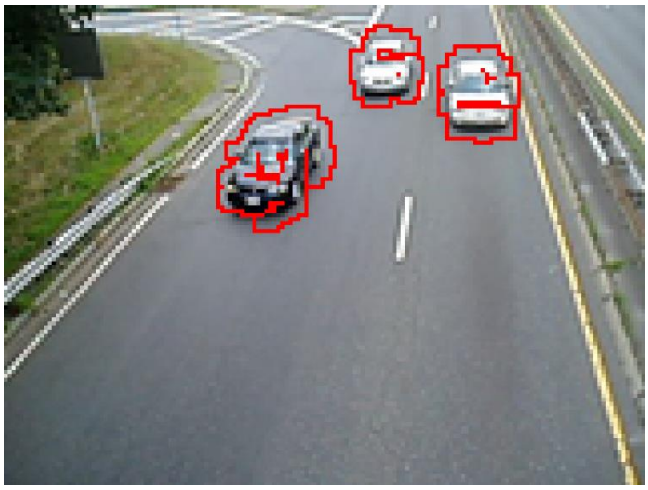

Figura 5-4. Trazado de fronteras difusas sobre fotograma.

Se observa que los resultados conseguidos son los esperados, pero estos distan mucho de ser válidos, las fronteras obtenidas carecen de claridad y precisión, de aquí que se haga referencia a ellas como difusas, lo que las hace poco viables para su uso en aplicaciones reales, además, al quedar huecos de píxeles en el interior de algunos objetos, estos son identificados también como partes de fronteras degradando el resultado final. Esto conduce a la necesidad de procesar estas fronteras en unas que enmarquen mejor al objeto detectado.

El código Matlab correspondiente a la obtención de las fronteras difusas se desarrolla en el anexo 7.

5.2 Procesamiento de fronteras

El procesamiento de fronteras persigue leer cada una de las fronteras difusas obtenidas en el punto anterior y convertirlas en un rectángulo que enmarque al objeto en movimiento.

Esto se consigue implementar de manera sencilla ya que únicamente será necesario, para cada frontera asociada a un objeto, encontrar los píxeles con valores extremos, los mínimos y los máximos, con los que se podrán trazar las nuevas fronteras rectangulares.

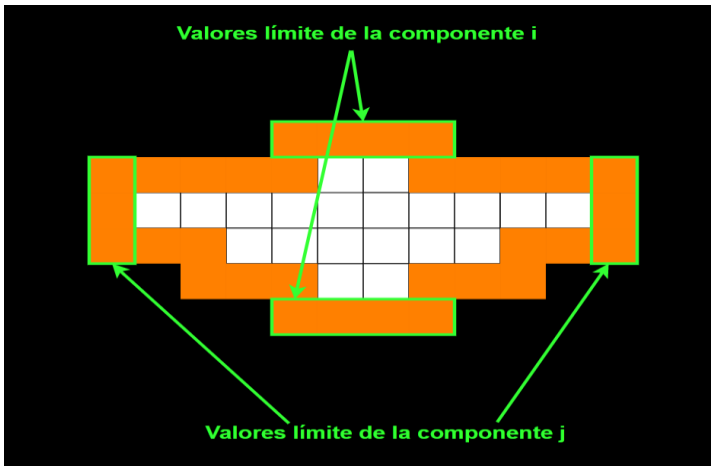

Figura 5-5. Valores límite de la frontera difusa.

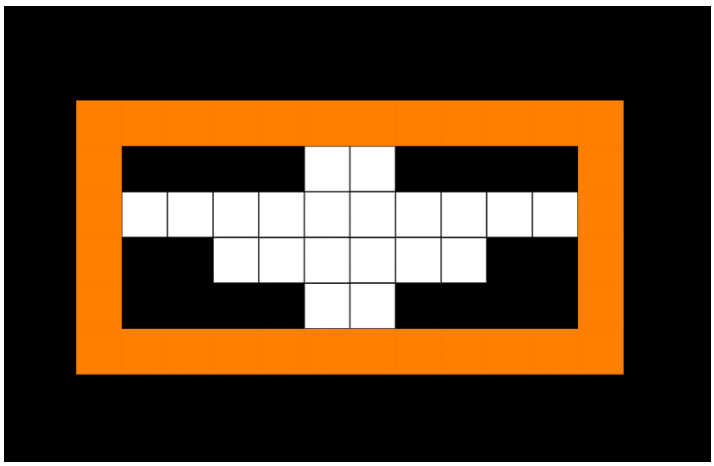

Figura 5-6. Resultado de empleo de los valores límites.

Aplicada esta función de procesado se podrán observar resultados mucho más óptimos a la hora de trazar estas nuevas fronteras sobre la secuencia de vídeo como se puede ver en la figura 5-7.

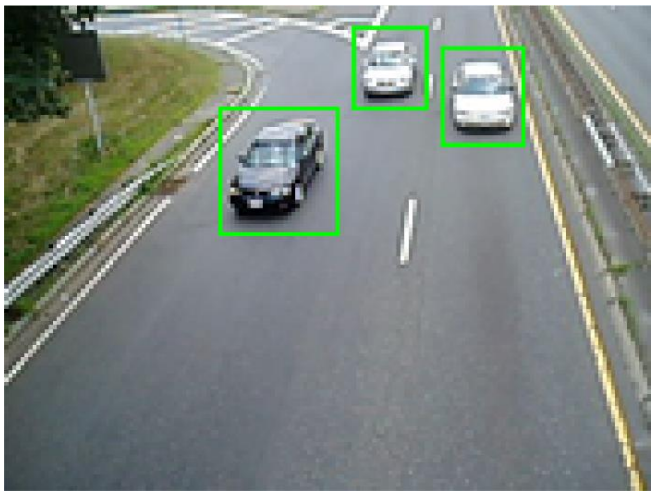

Figura 5-7. Fronteras procesadas sobre fotograma del vídeo.

El código Matlab correspondiente al procesamiento de fronteras se desarrolla en el anexo 8.

6 ELIMINACIÓN DE OBJETOS INTERIROES

La ciencia nunca resuelve un problema sin crear otros 10 más.

- George Bernanrd Shaw -

Con la obtención de las fronteras procesadas en el algoritmo ya era posible la observación de unos primeros resultados de la aplicación sobre la secuencia de vídeo, pero durante la ejecución de esta se contempló un error recurrente. Era común que en el interior de las fronteras que delimitaban al objeto apareciese otra frontera indicando la detección de un nuevo objeto.

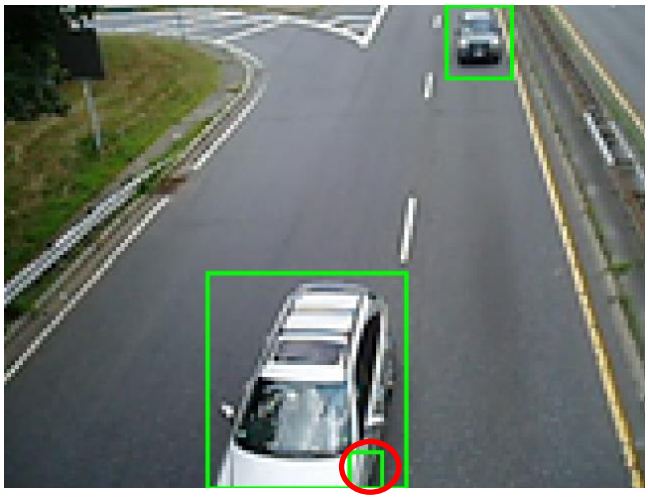

Figura 6-1. Objeto interior detectado.

Esto se debe a la ya comentada sensibilidad de la aplicación a cualquier imperfección, en este caso, que a pesar de los esfuerzos puestos en que los objetos quedasen perfectamente definidos para su diferenciación esta no se consiguiese en los pasos anteriores. Por ejemplo, a pesar del engordado de píxeles podía producirse que estos no se fusionasen con su objeto correspondiente creando uno nuevo. Para el caso concreto de la figura 6-1 su puede observar el la figura 6-2 esta imperfección.

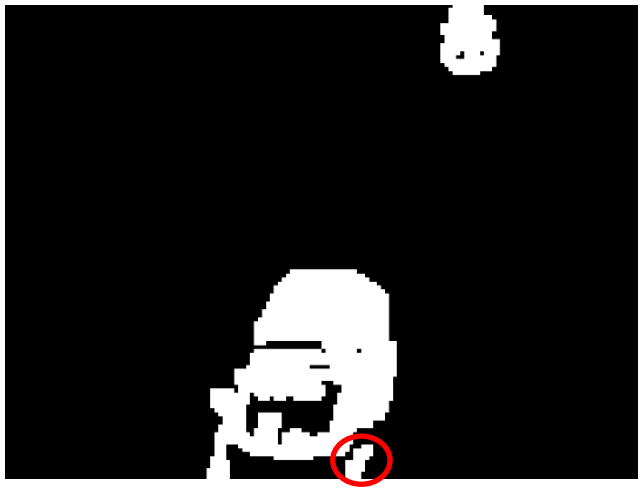

Figura 6-2. Objeto fragmentado a pesar de tratamiento.

Habiendo llegado a este punto se pudo optar por tomar uno de los siguientes caminos:

1- La recalibración de los parámetros de diseño vistos hasta ahora.

2- La construcción de una nueva función que corrigiese esta clase de errores.

Tras varios intentos de recalibración de los parámetros de diseño se llegó a la conclusión de que obtener mejores resultados de esta manera ya no resultaba posible, pues al menor cambio en estos aparecían nuevamente errores solventados con anterioridad. Volviendo al caso mostrado en las figuras anteriores se podría resolver particularmente el error engordando más aún los píxeles, pero esto provocaría que muchos otros objetos independientes se fusionasen como uno solo. Entonces se optó por la construcción de una nueva función.

La finalidad de esta nueva función sería la identificación de objetos que apareciesen totalmente o de forma parcial en el interior de otros objetos mayores.

Esto se conseguirá haciendo uso de los valores límites de las fronteras obtenidos en el apartado anterior. Se comprobará, para cada objeto si se cumplen al menos tres de las siguientes cuatro condiciones al compararlo con el resto de los objetos en el fotograma:

1- Frontera superior de un objeto se encuentra entre la frontera superior e inferior del objeto con el que se compara.
2- Frontera inferior de un objeto se encuentra entre la frontera superior e inferior del objeto con el que se compara.
3- Frontera izquierda de un objeto se encuentra entre los límites laterales del objeto con el que se compara.
4- Frontera derecha de un objeto se encuentra entre los límites laterales del objeto con el que se compara.

Se incluyen a continuación una serie de figuras mostrando a que hacen referencia estas condiciones.

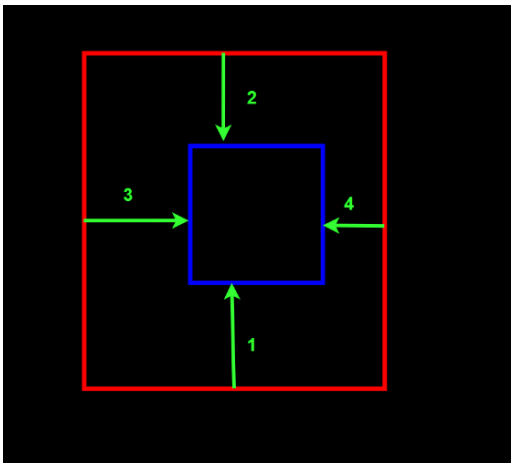

Figura 6-3 Objeto interior cumpliendo las 4 condiciones dadas.

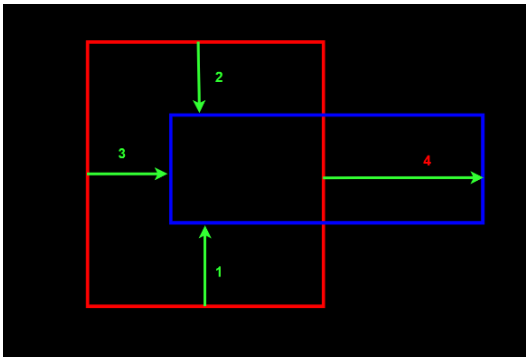

Figura 6-4. Objeto parcialmente interior cumpliendo 3 de las 4 condiciones dadas.

Cuando se den entonces al menos 3 de las condiciones entre un par de objetos, aquel con al menos 3 lados en el interior del otro objeto será considerado un error y eliminado del fotograma, además se reducirá el contador de objetos en el fotograma para poder llevar la cuenta de forma correcta. El resultado de aplicar esta función correctora se puede observar en la figura 6-5.

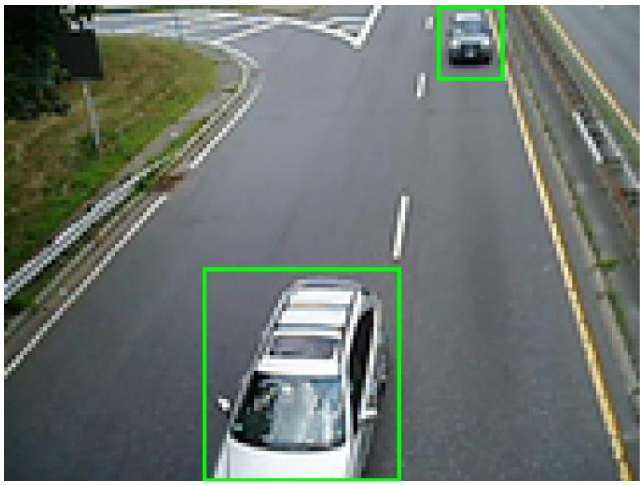

Figura 6-5. Resultado de aplicar la eliminación de objetos interiores.

El código Matlab correspondiente a esta función se encuentra en el anexo 9.

7 IDENTIFICACIÓN DE OBJETOS

No quiero creer. Quiero saber.

- Carl Sagan -

EN el punto del proyecto en el que nos encontramos ya se ha conseguido que en la secuencia de vídeo original todos los objetos en movimiento que aparecen sean rodeados por una frontera rectangular durante toda la secuencia. Pero hasta este momento no se había planteado la resolución del problema de identificación de objetos entre iteraciones. Es decir, cuando finaliza una iteración y se grafican los resultados comienza instantáneamente una nueva iteración que resolverá el problema de nuevo para un par de fotogramas distintos. Esto provoca que el movimiento de los objetos siga siendo detectado de una forma efectiva pero que la aplicación sea incapaz de reconocer si el objeto en movimiento localizado es alguno de los que ya había aparecido en la iteración anterior.

Se plantea entonces en este apartado el desafío de la identificación de estos objetos en movimiento, no solo para el fotograma actual, sino para la secuencia de vídeo completa. Para implementar esta funcionalidad se siguió una estrategia basada en la extracción de puntos característicos en los objetos detectados. La implementación de esta estrategia se divide en tres pasos:

1- Extracción de puntos característicos en los objetos detectados.

2- Búsqueda de coincidencias de los puntos característicos en iteraciones consecutivas para enlazar objetos que contengan a estos.

3- Comprobación final de errores.

7.1 Extracción puntos característicos

Cuando se habla de puntos característicos o singulares en el ámbito de la percepción se hace referencia a puntos en la imagen que presentan una baja autosimilitud.

La autosimilitud de un píxel es un parámetro que denota cuanto difiere este de aquellos que le rodean, si este difiere mucho resaltará más y por tanto será más singular. La obtención de este parámetro se lleva a cabo realizando la diferencia de sumas al cuadrado, SDC, en una vecindad en torno al píxel para el que queremos

calcular la autosimilitud.

A mayor SDC, menor será la autosimilitud y mayor la singularidad.

El objetivo que se persigue entonces es la búsqueda de puntos en la imagen, más concretamente en el interior de las fronteras que contienen a los objetos en la imagen, que presentan una baja autosimilitud.

La ecuación matemática que se emplea para el cálculo del SDC en referencia al valor de los píxeles es la siguiente:

$$SDC(\Delta_i, \Delta_j) = \sum_i \sum_j (f(i + \Delta_i, j + \Delta_j) - f(i,j))^2 \quad (7\text{-}1)$$

La ecuación 7-1 realiza el sumatorio de las diferencias al cuadrado entre el píxel para el que se calcula la autosimilitud y cada uno de los píxeles de la vecindad que se tiene en cuenta. Otro método para el cálculo del SDC se basa en el uso del tensor T.

$$SDC(\Delta_i, \Delta_j) = (\Delta_i, \Delta_j)\, T\, (\Delta_i, \Delta_j)^t \quad (7\text{-}2)$$

Este tensor se obtiene en base al valor del módulo del gradiente, tanto en la dirección x como en la y, de los píxeles en la vecindad para la que calculamos el SDC, ecuación 7-3.

$$T = \sum_i \sum_j \begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix} \quad (7\text{-}3)$$

Además de para el cálculo de la SDC, de este tensor T podemos obtener a partir de sus autovalores, λ_1 y λ_2, que tipo de singularidad tendrá el punto analizado. Como ya se había comentado la SDC que buscamos ha de ser elevada, entonces, si los dos autovalores tienen valores próximos a cero se considerará que el punto no es singular, si solo uno de los autovalores es próximo a cero mientras que el otro es muy elevado el punto analizado formará parte de algún borde, y si, por último, los dos autovalores son muy elevados se dirá que el punto pertenece a una esquina.

A la hora de buscar en nuestra aplicación los puntos característicos en lugar de emplear estas ecuaciones matemáticas se recurrirá a una función predefinida en Matlab que nos los dará de forma automática, pero sí que emplearemos el cálculo del tensor de forma que a cada punto singular se le adjuntará el tensor de su vecindad como seña de identidad, de forma que nos facilite el emparejamiento de puntos entre fotogramas en pasos posteriores.

Se construirá entonces una función Matlab que sea capaz de obtener los puntos singulares en el interior de las fronteras asociadas a cada uno de los objetos móviles detectados en el fotograma. Se obtendrán resultados de esta función fotogramas como el de la figura 7-1.

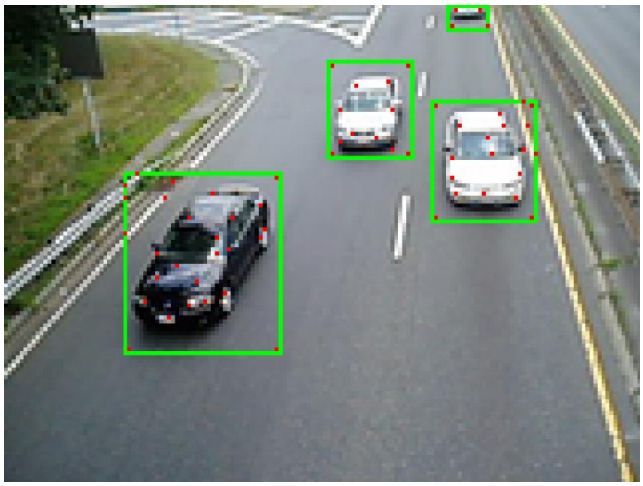

Figura 7-1. Puntos característicos extraídos en fotograma.

El código Matlab asociado a la obtención de puntos singulares se encuentra desarrollado en el anexo 8.

7.2 Emparejamiento de puntos singulares entre fotogramas

Tras la extracción de los puntos singulares de cada objeto en movimiento en una pareja de fotogramas consecutivos se comenzará un proceso mediante el cual se tratará de identificar los objetos en el segundo fotograma a partir de los puntos singulares de este y la información que ya se tiene de los objetos en el primero.

Los objetos en el primer fotograma se dan por identificados ya sea porque en el primer fotograma al no poder comparase con otro se les asocia un identificador de forma manual o porque estos ya han sido identificados por el algoritmo en la iteración anterior.

Se tratará entonces de buscar los puntos singulares de un objeto identificado en el primer fotograma en el interior de los objetos del segundo.

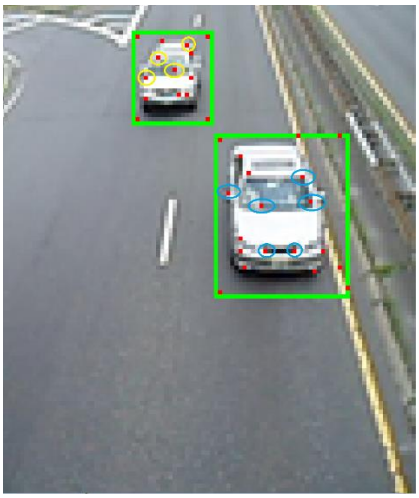

Figura 7-2. Puntos característicos en primer fotograma de pareja consecutiva.

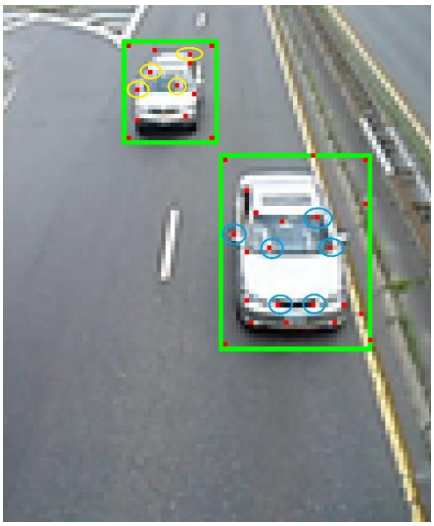

Figura 7-3. Puntos característicos en segundo fotograma de pareja consecutiva.

En las figuras anteriores, 7-2 y 7-3, se muestran los objetos en movimiento con sus respectivos puntos singulares en dos fotogramas consecutivos. Se observa que, aunque claramente los objetos en movimiento son los mismos los puntos singulares en ambos casos pueden diferir, ya que pueden aparecer en un mismo objeto nuevos puntos singulares o desaparecer puntos antiguos, aunque en otras circunstancias, como se observa en los casos remarcados sobre las figuras, algunos puntos son indiscutiblemente conjugados.

Se construirá entonces en la aplicación un sistema de identificación que otorgará a los objetos sus respectivos identificadores en base a un sistema de puntuaciones. Todos los puntos de cada objeto del primer fotograma serán comparados con todos los puntos de cada objeto en el segundo fotograma, el objeto en el segundo fotograma que presente más coincidencias a la hora de emparejar puntos, lo que se traduce en una puntuación más alta, se considerará como objeto parejo y por tanto se podrá identificar.

Este sistema de puntuaciones parte de comparar la información que tenemos de dos puntos, recordemos que en el apartado anterior se mencionaba que era empleado un tensor T en la vecindad del punto para asociarle a este punto cierta información que facilitase el emparejamiento. Esta información es almacenada en un vector c de

tres componentes normalizadas entre cero y uno, de tal forma que, al aplicar la ecuación 7-4 entre los dos puntos que están siendo comparados si se obtiene un valor próximo a uno significaría que los puntos podrían ser conjugados y que cuanto más nos alejemos de este valor menores serán las probabilidades de que lo sea.

$$c_{punto1} \cdot c_{punto2}{}' \tag{7-4}$$

Obtenido este valor se comparará con un parámetro umbral declarado de forma que si el resultado del producto no pasa el umbral el punto es descartado y en caso de que se consiga sobrepasar el punto será almacenado en una lista de potenciales candidatos. A la vez que un punto se almacena se mide la distancia existente entre los dos puntos que se han comparado.

Una vez se ha comparado un punto con todos en el fotograma siguiente se habrá obtenido como resultado una lista con aquellos que han conseguido pasar el umbral. Como solo se busca un punto de esta lista se iniciará un proceso de descarte que elimine a los demás, este descarte se realiza en base a la proximidad entre los puntos, por lo que el punto de la lista que haya pasado el umbral y que sea más próximo al punto que se está analizado pasará a considerarse su conjugado.

Esta distancia debe de cumplir como requisito una distancia mínima, ya que de encontrarse el conjugado a una gran distancia podría tratarse de un falso positivo. Esto se sabe por el análisis particular de la secuencia de vídeo donde los puntos singulares conjugados entre fotogramas no se alejan muchos píxeles entre sí. Entonces se declara esta distancia mínima como un nuevo parámetro de diseño de tal forma que si el punto la sobrepasa se considerará que este no era finalmente un conjugado y se eliminará dejando al punto analizado sin pareja.

Para los puntos que si han conseguido superar esta condición final se comprobará a que objetos identificados pertenecen y se les sumaran a estos puntos.

Cuando todos los puntos de un objeto han sido comparados con todos los puntos de cada objeto del fotograma siguiente se comparará la puntuación que ha obtenido cada objeto para realizar el emparejamiento. El objeto que más puntos tenga querrá decir que contiene la mayor cantidad de puntos conjugados con los de objeto analizado y pasará su identificador al del siguiente fotograma.

En este punto nos podemos encontrar con que ningún objeto resulte ser la pareja del analizado, todos tienen una puntuación de cero. Esto querrá decir que, salvo error, el objeto es nuevo en el fotograma, este no había aparecido con anterioridad. A este nuevo objeto se le asociará un nuevo identificador único que permita realizar su seguimiento en los posteriores fotogramas de la secuencia.

Los resultados de computar y graficar el algoritmo desarrollado se pueden observar en las figuras 7-4 y 7-5.

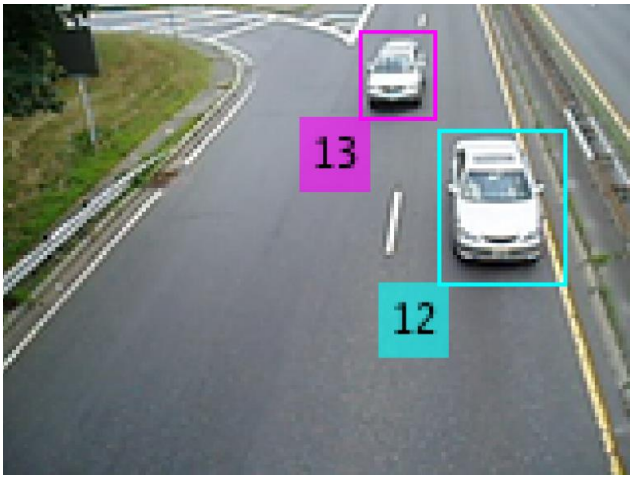

Figura 7-4. Identificación de objetos.

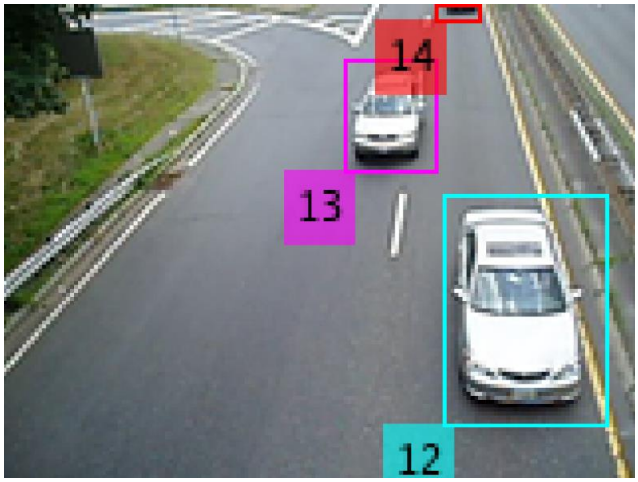

Figura 7-5. Identificación de objetos, aparición de nuevo objeto.

El código Matlab correspondiente a esta función se encuentra desarrollado en los anexos 9 y 11.

8 CORRECCIÓN DE ERRORES

Todo es teóricamente imposible, hasta que es hecho.

- Robert A. Heinlein -

EN el apartado anterior se finalizó la identificación de los objetos en movimiento y su seguimiento a lo largo de toda la secuencia de vídeo. Observando los resultados se comprobó un correcto funcionamiento del proceso, aunque se contemplaron eventualidades que degradaban ligeramente su comportamiento.

En la secuencia se podía ver como cuando al aparecer varios vehículos juntos al comienzo del carril durante breves instantes estos podían detectarse como un único objeto, esto en sí no suponía ningún problema ya que rápidamente se desvinculaban y eran reconocidos como objetos independientes. Pero en cambio sí que resultaba peor este caso cuando se incluía la identificación, pues, como se muestra en la siguientes figuras al desvincularse dos objetos que habían comenzado fusionados como uno solo ambos continúan la secuencia con el mismo identificador cuando se supone que este debería ser único para cada objeto detectado.

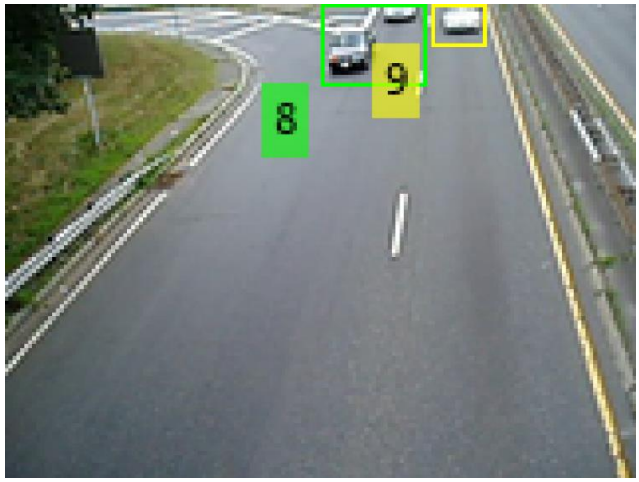

Figura 8-1. Dos objetos fusionados como uno reciben identificador.

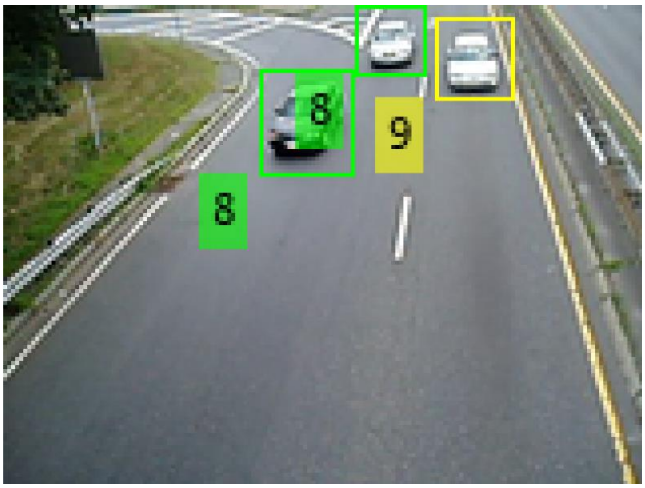

Figura 8-2. Objetos dejan de estar fusionados, pero comparten identificador.

Es entonces que se opta por la construcción de una nueva función para la corrección de estos casos especiales. La idea de esta es bastante sencilla, se encargará de comprobar en todo momento que los identificadores no se están compartiendo y en caso de encontrar alguno repetido le asociará a cada objeto uno diferente.

Los resultados de aplicar esta función se muestran en las figuras 8-3 y 8-4.

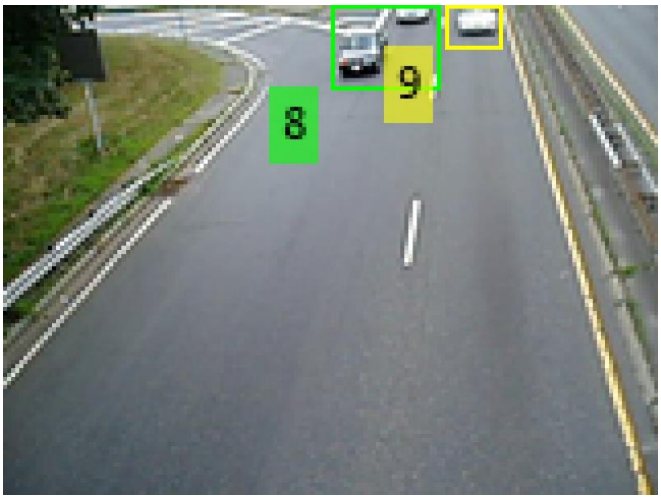

Figura 8-3. Objetos fusionados como uno reciben mismo identificador.

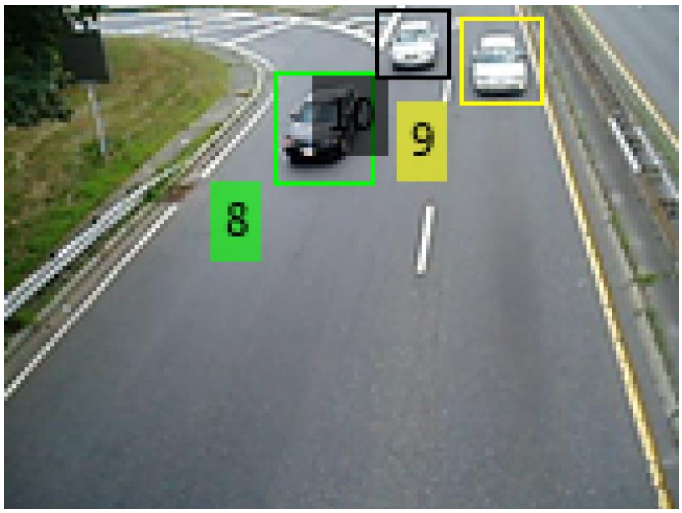

Figura 8-4. Objetos dejan de estar fusionados y reciben identificadores distintos.

Otra solución que reducía considerablemente este error sin la necesidad de implementar esta función fue cambiar el momento en el que se iniciaba la identificación, en vez de identificar los objetos automáticamente en el momento en el que aparecían al comienzo de la carretera se dejó un margen de filas hacia abajo del fotograma donde los objetos ya aparecen mejor definidos.

Figura 8-5. Recta a partir de la cual se comenzará la identificación.

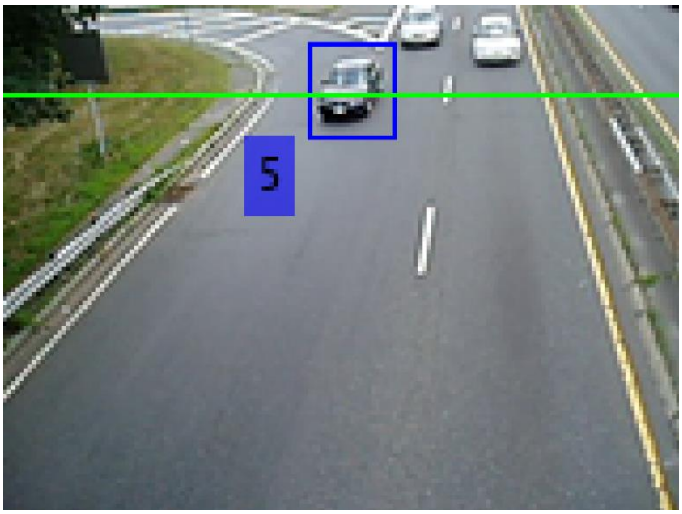

Figura 8-6. Identificación primer objeto que atraviesa la línea.

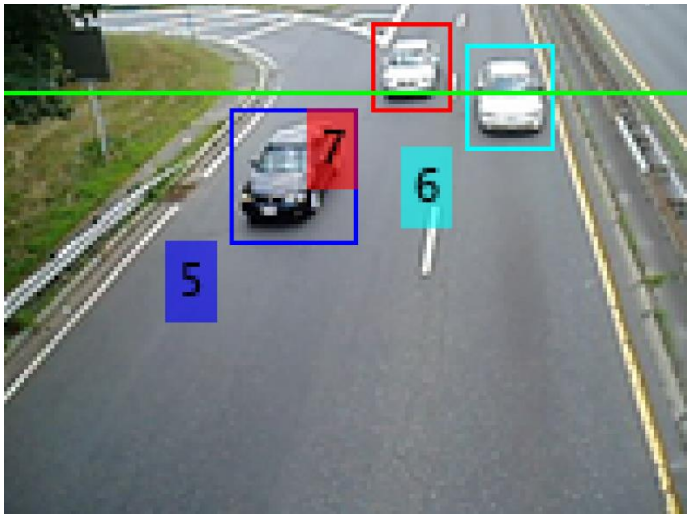

Figura 8-7. Identificación de múltiples objetos al atravesar la línea.

Esta solución fue con la que el algoritmo obtuvo mejores resultaos, más aún si se aplicaba en conjunto con la función de corrección de errores comentada. El problema es que esta consideración es muy dependiente del vídeo tratado, en este caso especial se sabe perfectamente de donde proceden los objetos y estos además aparecen siempre en el mismo sentido por tanto es posible ajustar esta línea umbral para la detección de los objetos y obtener los mejores resultados. Esto no es siempre así por lo que a la hora de usar este algoritmo con otros vídeos no se podrá tomar esta solución y se empleará la función para la corrección de identificadores.

El código Matlab correspondiente a este apartado se encuentra desarrollado en el anexo 10.

9 GENERACIÓN DE TRAYECTORIAS

La ciencia se trata de saber, la ingeniería de hacer.

- Henry Petroski -

A modo de aplicación añadida para el proceso de detección e identificado se decidió desarrollar además una serie de funciones con el fin de poder generar una trayectoria aproximada del desplazamiento que ha realizado cada objeto detectado en la secuencia de vídeo.

Con la incorporación de esta función se pretende por un lado que en la secuencia de vídeo sea visible la trayectoria que ha ido siguiendo el vehículo por medio de una estela que se va dibujando a su paso y por otro lado almacenar estas trayectorias para poder tratar o visualizar estas y obtener información de la secuencia.

La idea implementada para la obtención de estas trayectorias pasa por dos pasos, en primer lugar, en cada objeto de cada fotograma se debe obtener su centro de manera que cuando el mismo objeto sea identificado en el fotograma consecutivo se pueda trazar el camino que los une a ambos puntos, visualizándose el desplazamiento del punto central del objeto. Para llevar a cabo esta idea fue necesaria la construcción de dos nuevas funciones.

La primera se encarga únicamente de obtener los centros de los objetos detectados haciendo uso de las fronteras obtenidas con anterioridad. Se tomará el punto central como la coordenada marcada por la mitad de la altura de la frontera y la mitad de la base.

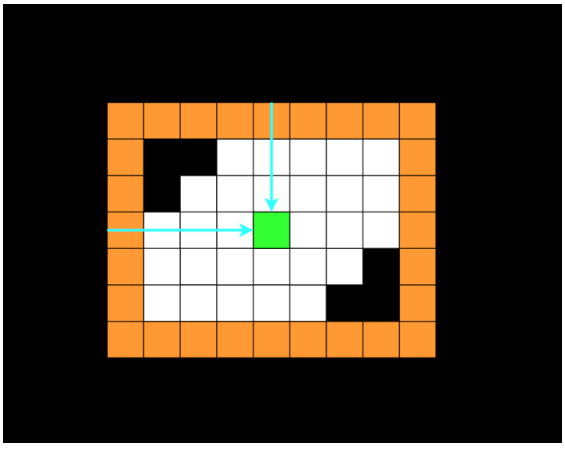

Figura 9-1. Trazado de centro del objeto.

El código Matlab correspondiente a la generación de centros se encuentra en el anexo 12.

Habiendo obtenidos los puntos centrales en todos los objetos móviles en dos fotogramas consecutivos el siguiente paso será para cada pareja de objetos que compartan el mismo identificador trazar los puntos que unen sus centros.

La mayor complejidad de este apartado fue conseguir que esta trayectoria que se va obteniendo al ir uniendo los centros del objeto entre los diferentes fotogramas fuese lo más suave posible. Se podrían haber tomado soluciones más sencillas para esta conexión de puntos como se muestran en las figuras 9-2 y 9-3.

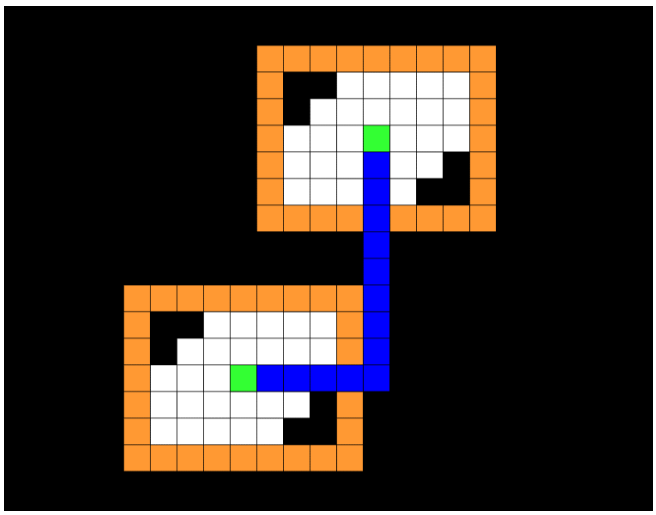

Figura 9-2. Unión de puntos simple, ejemplo 1.

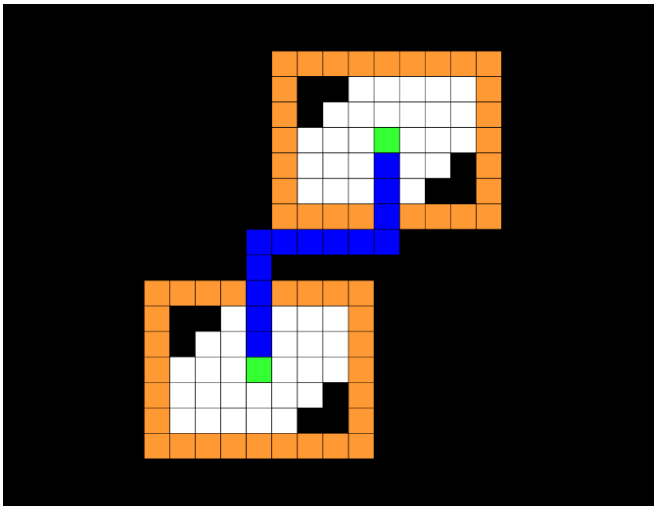

Figura 9-3. Unión de puntos simple, ejemplo 2.

Cualquiera de estas soluciones hubiese sido simple de implementar, pero las trayectorias obtenidas hubiesen resultado muy bruscas. En lugar de esto se optó por construir un método que consideraba las distancias entre los puntos, tanto en filas de píxeles como en columnas, para unir los puntos con una recta lo más diagonal posible.

En función de las distintas condiciones que se den, por ejemplo, mayor distancia en número de filas que en número de columnas o viceversa se creará una unión de pixeles dividida en pequeños segmentos como se muestra en la figura 9-4.

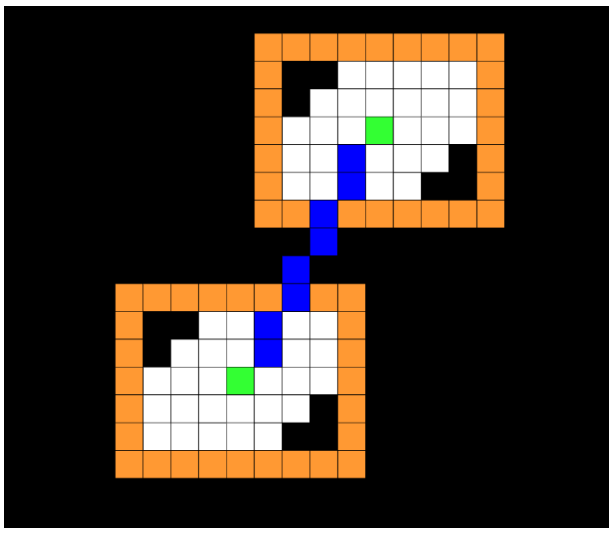

Figura 9-4. Unión suavizada.

De esta forma se obtiene una aproximación más realista al que habrá sido el desplazamiento de un objeto entre dos fotogramas consecutivos.

En el anexo 13 se desarrolla el código Matlab construido para la obtención de estos píxeles de trayectoria.

Construidas estas funciones se integrarían finalmente con las funciones encargadas de la visualización de resultados y se obtendrían fotogramas como los de las figuras 9-5 y 9-6 sobre el vídeo procesado.

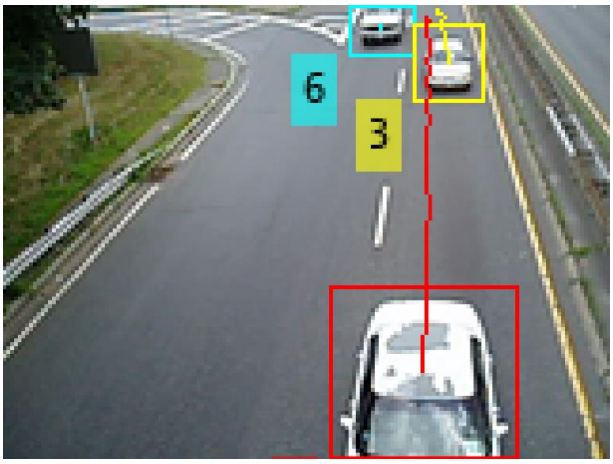

Figura 9-5. Fotograma resultado con trayectorias incorporadas, ejemplo 1.

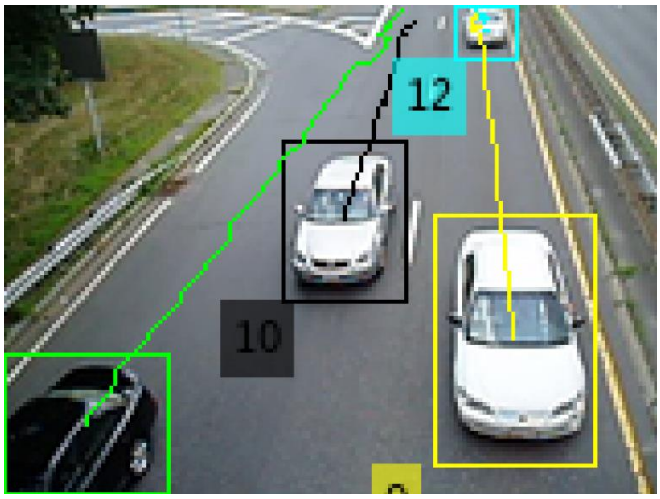

Figura 9-6. Fotograma resultado con trayectorias incorporadas, ejemplo 2.

Junto a la visualización de las trayectorias en la secuencia de vídeo se incorpora el graficado de estas ya sea en tiempo real o como resultados finales para poder observar cómo han sido los desplazamientos de todos los objetos a lo largo de la secuencia.

Para el vídeo empleado los gráficos resultados obtenidos fueron los siguientes:

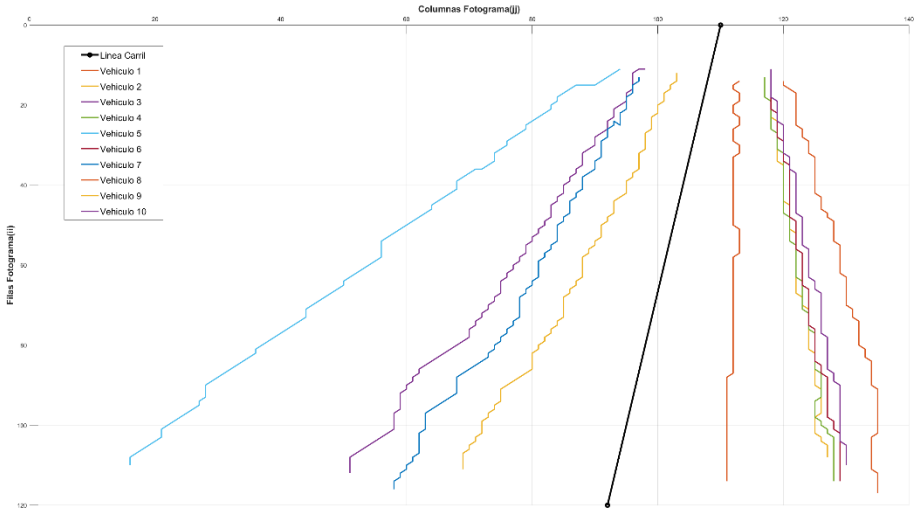

Figura 9-7. Graficación trayectorias de todos los vehículos.

En esta gráfica resultado se muestran todas las trayectorias obtenidas para los 10 vehículos que recorren la carretera a lo largo de la secuencia de vídeo.

Adicionalmente, para tratar cuantificar la bondad de las trayectorias obtenidas por el algoritmo se realizo una comparación entre estas y unas nuevas las cuales se simulaban que eran adquiridas por un experto humano.

Estas nuevas trayectorias se construyeron de forma similar a las originales, pero con ciertas variantes que pretendían simular las imprecisiones del ojo humano.

Por un lado, se supuso que el ojo humano sería incapaz de tomar la posición del objeto en cada uno de los fotogramas de la secuencia de vídeo dada la velocidad de estos, por lo que se redujo esta captura de puntos a uno por cada cuatro fotogramas, simulando así la menor velocidad de respuesta del ojo humano respecto a la velocidad de procesamiento del algoritmo.

Por otro lado, a la adquisición de la posición del objeto se le añadió una cierta componente de aleatoriedad. Mientras que el algoritmo para adquirir la posición del objeto en cada fotograma tomaba el píxel que ocupaba el centro geométrico de este, para simular la imprecisión del ojo humano se realizó de tal forma que se tomaba este centro más/menos un número aleatorio de píxeles que podían alejarlo, o no, de su centro exacto. Este número de píxeles aleatorios con los que podía variar la toma del punto era proporcional al tamaño del objeto de modo que cuanto mayor era el objeto más imprecisa resultaba la captura de su centro.

A continuación, se muestran algunas comparativas entre las trayectorias obtenidas por el algoritmo y las obtenidas por simulación del ojo humano.

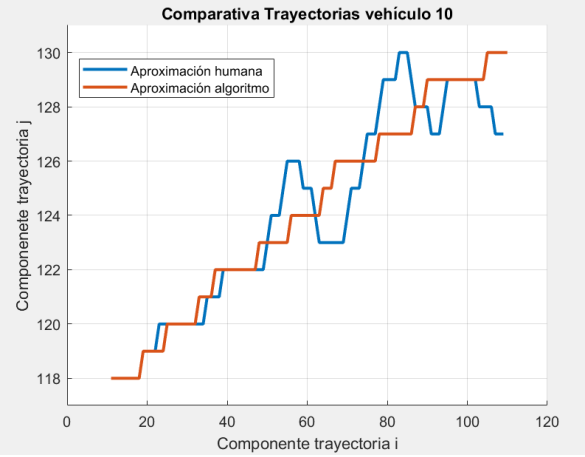

Figura 9-8. Comparativa trayectoria obtenida por algoritmo y trayectoria obtenida por humano, ejemplo 1.

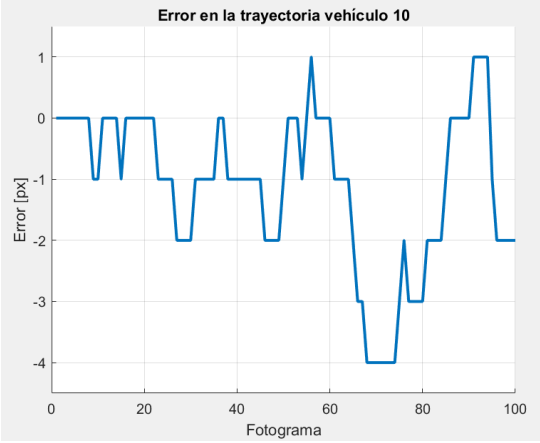

Figura 9-9. Error entre las trayectorias obtenidas por algoritmo y por humano, ejemplo 1.

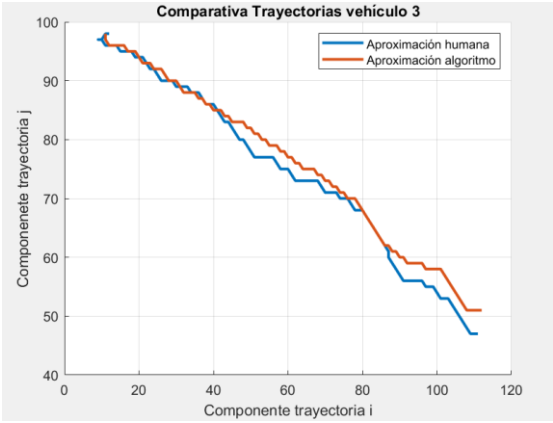

Figura 9-10. Comparativa trayectoria obtenida por algoritmo y trayectoria obtenida por humano, ejemplo 2.

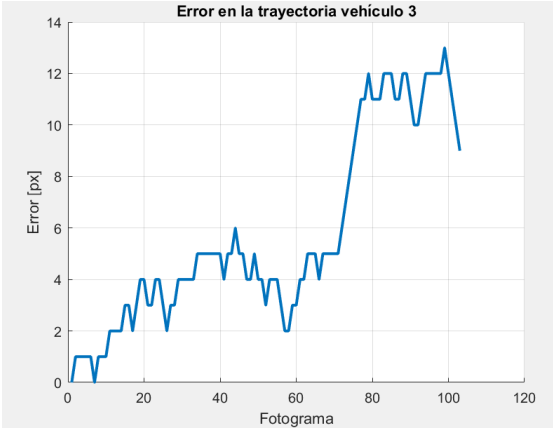

Figura 9-11. Error entre las trayectorias obtenidas por algoritmo y por humano, ejemplo 2.

Observando estas gráficas tomadas para el vídeo de la secuencia de tráfico se contempla como los errores que se calculan al comienzo de la secuencia de vídeo son casi nulos debido a que en este punto el vehículo acaba de aparecer y su tamaño es mínimo. A partir de este punto conforme el vehículo avanza hacia la cámara y su tamaño aumenta también se incrementa la imprecisión con la que se toma su centro.

Por último, hacer mención que debido a que el ojo humano toma los puntos cada cierto número de fotogramas puede suceder que no realice la adquisición de los fotogramas finales debido a que el objeto ha desaparecido de la secuencia de vídeo. Esto se comprueba ampliando el tramo final de las gráficas comparativas.

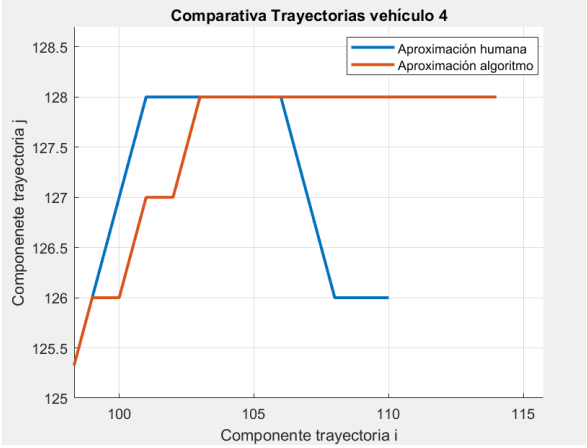

Figura 9-12. Muestras incompletas en la trayectoria obtenida por el experto humano.

Como no se ha podido obtener el punto final no solo se pierde este punto, sino que también se pierden los puntos que se obtendrían de conectar esta última muestra con la anterior.

En las gráficas de error no se incluyen estos tramos finales al no tener una pareja de puntos que comparar, en su lugar, para evitar problemas con el simulador de Matlab se realiza el cálculo del error hasta el último punto de la trayectoria obtenida con el ojo humano.

Se pueden observar las gráficas de error y comparativas de trayectorias de cada vehículo en el anexo número 14.

Una aplicación útil para esta funcionalidad, en el caso concreto de usarse para la detección de movimiento de vehículos en carretera como en el ejemplo desarrollado, podría ser el estudio del movimiento de vehículos en tramos determinados de carretera o la detección de cambios de carril.

10 CONCLUSIONES Y OBSERVACIONES FINALES

La ciencia se compone de errores, que, a su vez, son los pasos hacia la verdad.

- Julio Verne -

TRas finalizar la construcción del algoritmo se comenzaon a realizar diversas pruebas sobre este.

En primer lugar, se grabó un nuevo vídeo con una cámara de móvil desde una posición fija. Al tratar de procesar este vídeo se encontró que el programa se ralentizaba enormemente haciendo inviable su uso. Este problema se debe a que el algoritmo desarrollado en gran multitud de las funciones que emplea realiza lecturas de la imagen píxel a píxel, en el vídeo usado para la construcción del software este error no se presenciaba debido a que este contaba con una baja resolución, 120 píxeles de alto y 160 de ancho, pero el nuevo vídeo grabado pasaba a tener una resolución de 1920 píxeles de alto y 1080 de ancho. La diferencia de píxeles entre ambos casos es bastante abrupta:

- Número de píxeles para resolución 120x160: 19200 píxeles.
- Número de píxeles para resolución 1920x1080: 2073000 píxeles.

A pesar de la cantidad de cálculos que eran necesarios para este nuevo caso se dejó correr el algoritmo. En esta nueva secuencia de vídeo se enfocaba una mesa en la que por uno de sus lados aparecía un bolígrafo.

Los resultados que se obtuvieron en comparación con los de la secuencia de vídeo original no fueron óptimos. Para comenzar, antes de la aparición del objeto en movimiento se contemplaban pequeños objetos detectados donde en realidad no los había, lo que se estaban detectando eran nubes de píxeles que no habían sido limpiadas con los procedimientos creados, figura 10-1. Estas nubes podían proceder del tipo de iluminación a la que estaba sometido el vídeo.

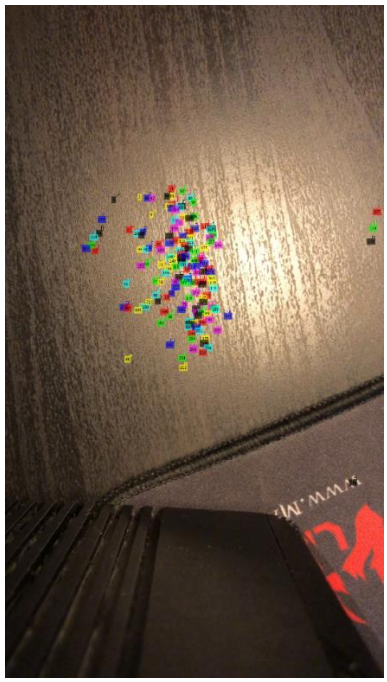

Figura 10-1. Píxeles de ruido detectados como objetos.

Además, cuando finalmente aparece el objeto a ser detectado este aparece segmentado en múltiples objetos, en lugar de uno solo.

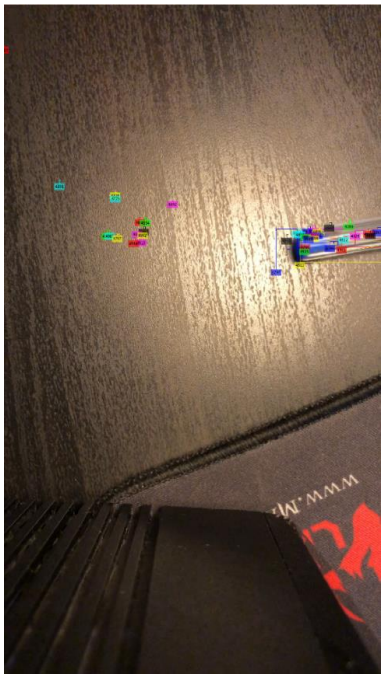

Figura 10-2. Objeto en movimiento segmentado en varios.

Todos estos defectos podrían haberse tratado de resolver recorriendo el algoritmo desde el principio y

reajustando los parámetros de diseño hasta alcanzar resultados que se consideren óptimos.

Tras esta primera prueba fallida se optó por la búsqueda de otro vídeo sobre el que probar el algoritmo, en esta ocasión se empleó un vídeo con una resolución del doble del vídeo original, 240 píxeles de alto y 320 de ancho, en esta secuencia de vídeo a cámara fija se observa un paseo por el que circulan diferentes personas.

Al cargar el vídeo nuevamente se observa una ralentización al tener esta resolución más píxeles, pero no tan alarmante como en el primer ejemplo. Los resultados en esta secuencia de vídeo son bastante mejores al primer caso como se muestra en las siguientes figuras:

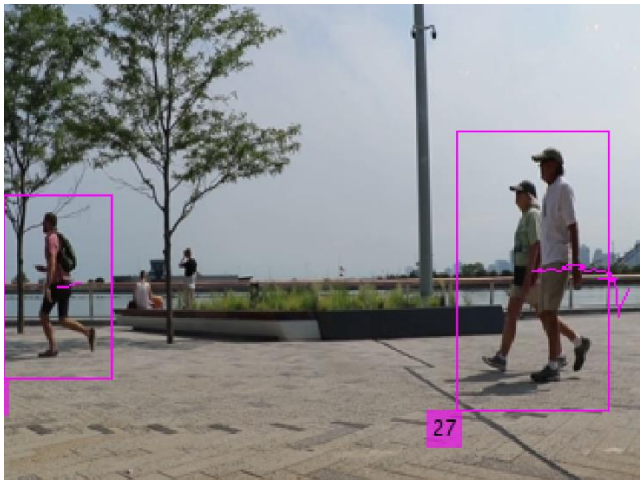

Figura 10-3. Resultados vídeo paseo, ejemplo 1.

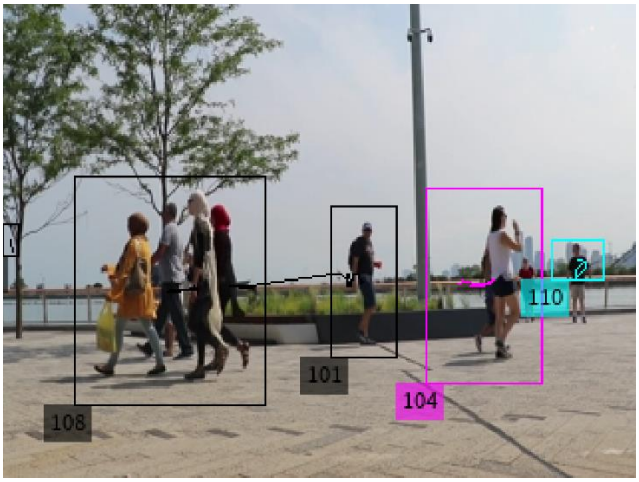

Figura 10-4. Resultados vídeo paseo, ejemplo 2.

Figura 10-5. Resultados vídeo paseo, ejemplo 3.

Aunque la detección de objetos parece funcionar mejor, se encuentran otros problemas nuevamente debido a la calibración de los parámetros a la hora de identificar los objetos. Estos fallos provocaban la fácil pérdida momentánea de objetos con lo que se reiniciaba su identificación, dándose al objeto perdido un nuevo número de identificación en el siguiente fotograma en el que se detectaba, lo que a su vez provocaba que su trayectoria recorrida desapareciese de la secuencia.

Como conclusiones se tienen entonces que el algoritmo diseñado en el caso de que resultase de importancia la velocidad de cálculo, como, por ejemplo, aplicaciones en tiempo real, debería usarse únicamente sobre vídeos de baja resolución. De no ser así y emplearse el algoritmo para analizar secuencias de vídeo donde no se presta atención a la velocidad con la que opera el algoritmo sí que podría emplearse para mayores resoluciones otorgándole el tiempo necesario para realizar los cálculos.

Por otro lado, para su uso apropiado siempre será importante realizar pruebas iniciales que permitan calibrar el algoritmo ya que este es extremadamente sensible a cambios como se ha ido viendo en el desarrollo. Antes de ponerlo en funcionamiento sería necesario volver a seleccionar todos los parámetros de diseño obtenidos a lo largo del proyecto, pues estos permitirán una mejor adaptación para las condiciones de iluminación, tipos de objeto, etc.

Incluyendo un breve resumen, los parámetros de diseño ajustables que se pueden encontrar a lo largo de las diversas funciones del proyecto son los siguientes:

1- Umbral para la transformación de imagen en diferencia a imagen binaria.
2- Tamaño de ventana para aplicar el método morfológico de erosión.
3- Tamaño de ventana para aplicar el método morfológico de acreción.
4- Puntuación mínima, resultado del producto de vectores de información, para realizar emparejamiento de puntos conjugados.
5- Distancia máxima entre píxeles para realizar emparejamiento de puntos conjugados.

Una alternativa para reducir los problemas de carga computacional y de identificación podría ser combinar la aplicación desarrollada, en concreto el apartado de detección de objetos, con técnicas de seguimiento de objetos más novedosas como ‘Camshift’ y ‘Meanshift’. Estas técnicas para el seguimiento presentan una pequeña tara debido a que necesitan en un primer momento que se le indique el objeto que deben de seguir para obtener su distribución de color, este paso inicial que no realizan los algoritmos, reduciendo su automatización, podría ser solventado empleándose el proyecto desarrollado como paso inicial.

En el ejemplo desarrollado de los vehículos, se podría establecer una pequeña región de la imagen en el carril por donde circulan para realizar la detección y dar paso entonces a los métodos ‘Camshift’ y ‘Meanshift’ para realizar el seguimiento [2], lo que agilizaría los cálculos del programa.

11. ANEXOS

11.1 Estructura del algoritmo.

Este código se encuentra implementado en el archivo de proyecto llamado 'Main_core.m', este será el encargado de invocar a todos los demás ficheros cuando la función que desempeñan sea necesaria.

El algoritmo Matlab se ha planteado de forma que nada más iniciarse la ejecución realice la carga del vídeo que tendrá que tratar y extraiga sus parámetros, funcionalidad desarrollada en el anexo 2.

Tras esto se entraría en una estructura compuesta por un bucle 'for' que tendrá tantas iteraciones como fotogramas el vídeo, a este bucle se accede a partir del segundo fotograma ya que para poder realizar la diferencia de imágenes se necesitará siempre la existencia de un fotograma anterior.

Una vez dentro de este bucle se irán invocando las diferentes funciones de procesamiento construidas para el proyecto cuyo código será desarrollado en los siguientes anexos y cuyos resultados se irán almacenando en una estructura de datos que ha sido bautizada como 'FrameData'.

```
name_video = 'traffic.mj2';
[Video] = LoadVideo(name_video);
```

```
for n = 2 : Video.n_Fotos

    FrameData(n-1).n = n-1;
    trayectorias.n = n;

    %% Cálculo diferencia de fotogramas
    FrameData = ImageDiffer(Video,FrameData,n);

    %% Limpieza de motas de pixeles
    FrameData = DeletePixels(Video,FrameData,n);

    %% Engordado de objetos, facilitar diferenciación
    FrameData = FattenItems(Video,FrameData,n);

    %% Cuenta de objetos en Frame
    FrameData = CountItems(Video,FrameData,n);

    %% Obtención fronteras (Difusas)
    FrameData = FuzzyBorders(Video,FrameData,n);

    %% Procesador de fronteras (obtenemos objeto encuadrado)
    FrameData = ProcessBorders(FrameData,n);

    %% Eliminar objetos dentro de otros objetos
    FrameData = DeleteInnerItems(FrameData,n);
    FrameData = PlotProcessedBorders(FrameData,n);

    %% Obtención puntos sigulares objetos
    FrameData = SingularPoints(Video,FrameData,n);

    %% Generacion centros de objetos
    FrameData = GenerateCenters(FrameData,n);

    %% Reconocimiento entre frames
    if n >2
        [FrameData,Global_identifier] = pointsPairs(FrameData,n,Global_identifier);
        [FrameData, Global_identifier] = CheckErrors(FrameData,n,Global_identifier);
        [FrameData,trayectorias] = PlotBordersID(FrameData,n,trayectorias);
    end
end
```

11.2 Código para la carga de vídeo y extracción de parámetros.

Este código se encuentra implementado en el archivo de proyecto 'LoadVideo.m', este archivo es empleado como una función Matlab que se invoca desde el fichero principal. Para que esta función pueda funcionar deberá recibir como parámetro de entrada el directorio con la ubicación del vídeo a cargar, en el caso especial de usar un vídeo de la librería de Matlab bastará con dar el nombre del vídeo.

Esta función se encargará entonces de leer este vídeo y almacenarlo en una variable de Matlab de forma que pueda ser usado como una estructura de datos. Esto permitirá la extracción de parámetro importantes para el desarrollo del proyecto. Estos parámetros son:

- Número de filas de cada fotograma del vídeo.
- Número de columnas de cada fotograma del vídeo.
- Número de canales de color.
- Número total de fotogramas.

Tras su extracción tanto los parámetros extraídos como el vídeo en sí serán almacenados en una estructura de datos que se llamará 'VideoData' para su uso a lo largo del algoritmo.

```
function [VideoData] = LoadVideo(name_video)

%% Almacenado vídeo
video_load = VideoReader(name_video);
video = read(video_load);

%% Extracción de parámetros
M = size(video,1);
N = size(video,2);
color_channels = size(video,3);
frames = size(video,4);

%% Almacenado variables en estructura de datos
VideoData.video = video_load;
VideoData.M = M;
VideoData.N = N;
VideoData.color_channels = color_channels;
VideoData.n_Fotos = frames;

end
```

11.3 Diferencia de imágenes.

Este código se encuentra implementado en el archivo de proyecto 'ImageDiffer.m', este archivo es empleado como una función Matlab que se invoca desde el fichero principal. Para que esta función pueda funcionar recibirá como parámetros de entrada la estructura de datos que contiene tanto el vídeo como sus propiedades, estructura 'Video', la estructura de datos donde se van almacenando todos los datos obtenidos 'FrameData' y por último un parámetro que nos indica en que iteración nos encontramos.

La función comienza comprobando el número de canales de color del vídeo, de forma que si tiene tres significará que se trata de un vídeo RGB o en caso contrario de un vídeo en escala de grises. En realidad, podrían comprobarse otros casos al existir la posibilidad de recibir un vídeo con sus fotogramas en formato HSV los cuales también emplean 3 canales de color u otros con formato CMYK los cuales emplean 4 canales de color, pero por simplicidad estos se han ignorado.

Tras la comprobación la función extraería los dos fotogramas consecutivos correspondientes a la iteración en la que se encuentra, si nos encontramos en la iteración 'n' extraerá el fotograma 'n' y el fotograma 'n-1', y los convertirá a escala de grises en el caso de ser fotogramas RGB. Adicionalmente estos son almacenados en la estructura de resultados para posibilitar su acceso futuro.

```
function [FrameData] = ImageDiffer(Video,FrameData, n)

    if Video.color_channels == 3

        I1_RGB = read(Video.video, n-1);
        I2_RGB = read(Video.video, n);

        I1_gray = rgb2gray(I1_RGB);
        I2_gray = rgb2gray(I2_RGB);

        FrameData(n-1).images.I1_RGB = I1_RGB;
        FrameData(n-1).images.I2_RGB = I2_RGB;
        FrameData(n-1).images.I1_gray = I1_gray;
        FrameData(n-1).images.I2_gray = I2_gray;

    else

        I1_gray = read(Video.video, n-1);
        I2_gray = read(Video.video, n);

        FrameData(n-1).images.I1_gray = I1_gray;
        FrameData(n-1).images.I2_gray = I2_gray;

    end
```

Tras esto se muestra cómo se obtendría la imagen en diferencia y la imagen binaria según el método tradicional, se restan los valores píxel a píxel de cada fotograma y se aplica un umbral para obtener la imagen binaria, pero como ya se ha comentado al obtener peores resultados este método no fue empleado finalmente.

```
I_difference = I2 - I1;
I_proc= I_difference > 20;

FrameData(n-1).images.I_diff = I_difference;
FrameData(n-1).images.I_proc = I_proc;
```

En su lugar se empleó la diferencia entre los módulos de gradiente de cada imagen y se aplicó otro umbral para

la obtención de la imagen binaria.

```
[Gm1,~] = imgradient(I1,'sobel');
[Gm2,~] = imgradient(I2,'sobel');
G_diff = Gm2 - Gm1;

umbral = -90;
I_binary = G_diff  < umbral;
```

11.4 Erosión de píxeles.

Este código se encuentra implementado en el archivo de proyecto 'DeletePixels.m', este archivo es empleado como una función Matlab que se invoca desde el fichero principal. Para que esta función pueda funcionar recibirá como parámetros de entrada la estructura de datos que contiene tanto el vídeo como sus propiedades, estructura 'Video', la estructura de datos donde se van almacenando todos los datos obtenidos 'FrameData' y por último un parámetro que nos indica en que iteración nos encontramos.

La función comienza cargando la imagen binaria obtenida como resultado de la función anterior 'ImageDiffer.m', esta se encuentra almacenada en la estructura de datos 'FrameData'.

Tras esto se declara el tamaño máximo que queremos para la ventana de filtrado de píxeles dándole un valor a la variable 'max_size_window', el valor que le damos a esta variable no es exactamente el tamaño de un lado de la ventana, el lado de esta se obtendría con la siguiente ecuación:

$$ladoVentana = (maxSizeWindow * 2) + 1 \tag{11-1}$$

De tal forma que, si a la variable le damos en la ecuación 11-1 un valor de 1 la ventana de mayor tamaño que se usará será 3x3, si le damos un valor de 2 la ventana será 5x5 y así sucesivamente.

Escogido el valor de la ventana de mayor tamaño entraremos en el interior de una estructura de triple bucle 'for'. El primer bucle lo que hará será incrementar el tamaño de la ventana de limpieza cada vez que se termina de leer la imagen, de esta forma se elimina en el primer barrido los píxeles aislados y en las siguientes iteraciones las motas cada vez mayores. Los dos bucles siguientes se corresponden con la estructura clásica de doble bucle 'for' para la lectura de imágenes píxel a píxel, el primero lee las filas y el segundo las columnas. Los parámetros que indican los límites del fotograma se extraen de las propiedades del vídeo que habían sido almacenados en la estructura de datos 'Video'. Entonces se comenzará con la lectura de píxeles y solo se detendrá la función para analizar píxeles con valor de 1.

```
function [FrameData] = DeletePixels(Video,FrameData,n)

I = FrameData(n-1).images.I_difference;
max_size_window = 2;

for size_window = 1:max_size_window
    for ii = 1 : Video.M
        for jj = 1 : Video.N

            if I(ii,jj) == 1
                caso = 0;
```

La mayor complejidad en el desarrollo de esta función consistió en conseguir que esta operase independientemente del tamaño de la ventana, recordemos que Matlab nos devuelve un error fatal que detiene el proceso cuando se trata de leer un píxel para unas coordenadas fuera de los límites de la imagen, es decir, un píxel inexistente. La razón de que se produzca este error es el análisis de un píxel sobre un borde de la imagen o próximo a estos pues al colocar la ventana sobre estos los límites de la ventana quedan fuera de los bordes de la imagen produciendo este error. La solución que se dio a este problema fue la construcción de ventanas adaptativas cuando se diesen los casos especiales de que el píxel a analizar se encontrase sobre un borde u esquina, de forma que esta ventana se recorta sin exceder nunca los límites de la imagen, esta idea se representa gráficamente en las siguientes figuras.

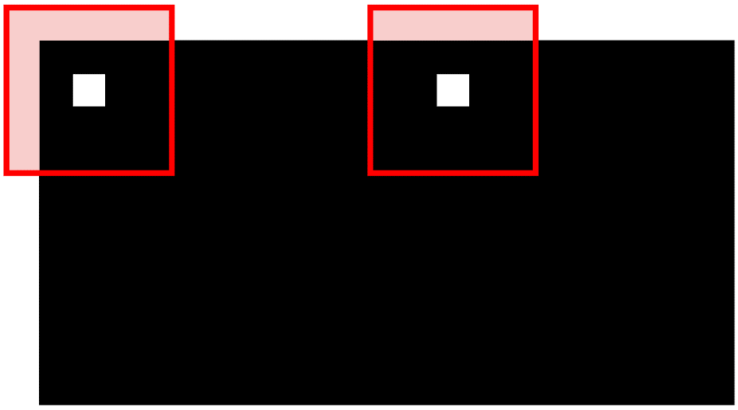

Figura 11-1. Ventanas desbordan límites de la imagen.

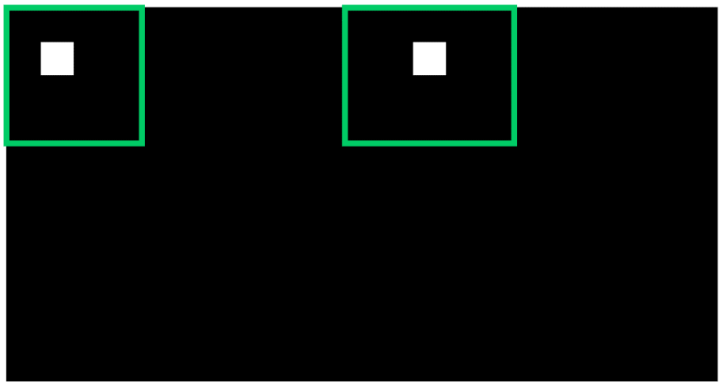

Figura 11-2. Ventanas adaptativas, no desbordan límites de la imagen.

Se inicializa entonces un variable ‘caso’ que indicará si nos encontramos en alguno de estos casos especiales o no, los valores que puede tomar esta variable son los siguientes:

0- No se produce caso especial.
1- Píxel en borde superior.
2- Píxel en borde inferior.
3- Píxel en borde izquierdo.
4- Píxel en borde derecho.
5- Píxel esquina superior izquierda.
6- Píxel esquina superior derecha.
7- Píxel esquina inferior izquierda.
8- Píxel esquina inferior derecha.

La variable parte inicializada a 0 con lo que partimos en el caso general y tras esta inicialización aparecerán 8 sentencias a las cuales sólo se podrá acceder cuando se den las condiciones de caso especial.

```
% 1º Borde superior, no esquina
if (ii <= size_window) && (jj > size_window) && ( jj <= Video.N-size_window)
    margin = size_window - ii;
    if margin < 1
        margin = 1;
    end
    window = I( margin : ii+size_window , jj-size_window : jj+size_window);
    caso = 1;
end

% 2º Borde inferior, no esquina
if (ii >= Video.M-size_window) && (jj > size_window) && ( jj <= Video.N-size_window)
    margin = size_window + ii;
    if margin > Video.M
        margin = Video.M;
    end
    window = I( ii-size_window : margin, jj-size_window : jj+size_window);
    caso = 2;
end

% 3º Borde izquierdo, no esquina
if (jj <= size_window) && (ii > size_window) && (ii <= Video.M-size_window)
    margin = size_window - jj;
    if margin < 1
        margin = 1;
    end
    window = I( ii-size_window : ii+size_window, margin : jj+size_window);
    caso = 3;
end

% 4º Borde derecho, no esquina
if (jj >= Video.N-size_window) && (ii > size_window) && (ii <= Video.M-size_window)
    margin = size_window + jj;
    if margin > Video.N
        margin = Video.N;
    end
    window = I( ii-size_window : ii+size_window, jj-size_window : margin);
    caso = 4;
end

% 5º Esquina superior izquierda
if (ii <= size_window) && (jj <= size_window)
    margin1 = size_window - ii;
    margin2 = size_window - jj;
    if margin1 < 1
        margin1 = 1;
    end
    if margin2 < 1
        margin2 = 1;
    end
    window = I(margin1 : ii+size_window,  margin2:jj+size_window);
    caso = 5;
end
```

```
% 6º Esquina superior derecha
if (ii <= size_window) && (jj > Video.N-size_window)
    margin1 = size_window - ii;
    margin2 = size_window + jj;
    if margin1  < 1
        margin1 = 1;
    end
    if margin2 > Video.N
        margin2 = Video.N;
    end
    window = I(margin1 : ii+size_window, jj-size_window:margin2);
    caso = 6;
end

% 7º Esquina inferior izquierda
if (ii > Video.M-size_window) && (jj <= size_window)
    margin1 = size_window + ii;
    margin2 = size_window - jj;
    if margin1  > Video.M
        margin1 = Video.M;
    end
    if margin2 < 1
        margin2 = 1;
    end
    window = I(ii-size_window : margin1, margin2 : jj+size_window);
    caso = 7;
end

% 8º Esquina inferior derecha
if (ii > Video.M-size_window) && (jj > Video.N-size_window)
    margin1 = size_window + ii;
    margin2 = size_window + jj;
    if margin1  > Video.M
        margin1 = Video.M;
    end
    if margin2 > Video.N
        margin2 = Video.N;
    end
    window = I(ii-size_window : margin1, jj-size_window : margin2);
    caso = 8;
end
```

Se explicará a continuación en detalle uno de los casos, por ejemplo, el caso 7.

Se comienzan comprobando dos condiciones sobre la posición del píxel. La primera condición comprueba que la diferencia entre el límite de la imagen y el tamaño de la mitad de la ventana son menores que la fila en la que se encuentra el píxel a analizar, esto indicaría que al colocar la ventana sobre el píxel se excederían las filas de esta, por tanto, nos encontramos en la situación de píxel sobre borde inferior. Además, se incluye una segunda condición que comprueba que el tamaño de la mitad de la ventana empleada sea mayor que la columna en la que se encuentra este píxel. Si se cumplen ambas condiciones no solo se verifica que el píxel se encuentra sobre una zona próxima al borde inferior, sino que también lo está en una zona próximo al borde izquierdo, por lo que se considera el caso de esquina inferior izquierda.

Entonces se establecen nuevos márgenes para la ventana por sus lados inferior e izquierdo para que no rebasen los límites.

```
% 7º Esquina inferior izquierda
if (ii > Video.M-size_window) && (jj <= size_window)
    margin1 = size_window + ii;
    margin2 = size_window - jj;
    if margin1  > Video.M
        margin1 = Video.M;
    end
    if margin2 < 1
        margin2 = 1;
    end
    window = I(ii-size_window : margin1, margin2 : jj+size_window);
    caso = 7;
end
```

Si representamos gráficamente las variables mostradas en el fragmento del código quedaría el resultado que se observa en la figura 11-3.

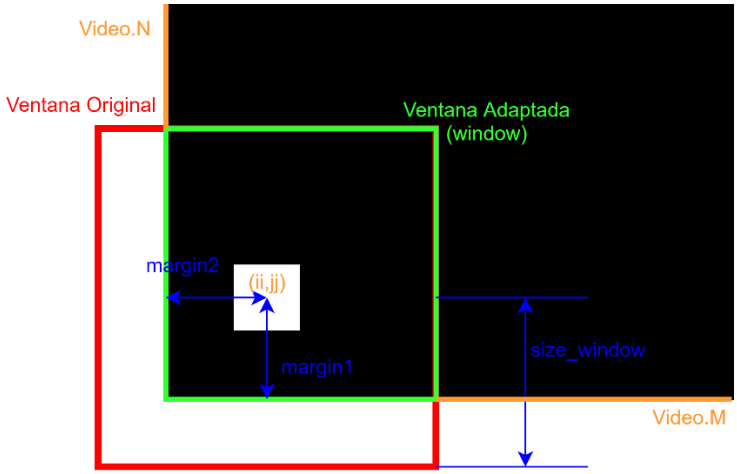

Figura 11-3. Conversión ventana tamaño fijo a ventana tamaño adaptativo.

En el caso de que no se entrase en ninguno de los casos especiales la ventana quedaría en sus valores por defecto y se produciría el filtrado interior de los píxeles.

```
if caso == 0
    window = I(ii-size_window : ii+size_window, jj-size_window : jj+size_window);
end

[mm,nn] = size(window);
min(window(:,1))
if caso == 0
    if sum(window(:,1)) + sum(window(:,nn)) +sum(window(1,:)) + sum(window(mm,:)) == 0
        I(ii-size_window : ii+size_window, jj-size_window : jj+size_window) = 0;
    end
elseif caso == 1
    if sum(window(:,1)) + sum(window(:,nn))  + sum(window(mm,:)) == 0
        I( margin : ii+size_window , jj-size_window : jj+size_window) = 0;
    end
elseif caso == 2
    if sum(window(:,1)) + sum(window(:,nn)) + sum(window(1,:)) == 0
        I( ii-size_window : margin, jj-size_window : jj+size_window) = 0;
    end
elseif caso == 3
    if sum(window(:,1)) + sum(window(:,nn)) + sum(window(mm,:)) == 0
        I( ii-size_window : ii+size_window, margin : jj+size_window)=0;
    end
elseif caso ==4
    if sum(window(:,1)) +sum(window(1,:)) + sum(window(mm,:)) == 0
        I( ii-size_window : ii+size_window, jj-size_window : margin)=0;
    end
elseif caso ==5
    if  sum(window(:,nn))  + sum(window(mm,:)) == 0
        I(margin1 : ii+size_window,  margin2:jj+size_window) = 0;
    end
elseif caso ==6
    if sum(window(:,1))+ sum(window(mm,:)) == 0
        I(margin1 : ii+size_window, jj-size_window:margin2)=0;
    end
elseif caso ==7
    if  sum(window(:,nn)) +sum(window(1,:)) == 0
        I(ii-size_window : margin1, margin2 : jj+size_window)=0;
    end
elseif caso == 8
    if sum(window(:,1)) +sum(window(1,:))  == 0
        I(ii-size_window : margin1, jj-size_window : margin2)=0;
    end
end
```

Se concluye realizando la comprobación para eliminar el píxel, esto se lleva a cabo sumando el valor de todos los píxeles sobre el perímetro exterior de la ventana, si la suma total vale 0 significaría que se trata de un píxel o mota aislada y se eliminaría, para los casos especiales la comprobación se realiza sin considerar los límites de la imagen como bordes del perímetro de forma que en vez de comprobar los 4 lados de la ventana solo se comprueban los 3 lados o 2, dependiendo del caso, que no tocan los límites de la imagen. Esta aclaración se debe a que al adaptar la ventana el propio píxel que está siendo analizado pude pasar a formar parte del perímetro de la ventana por lo que nunca sería filtrado aun estando aislado.

Finalmente se almacena la imagen filtrada en la estructura de datos ‘FrameData’.

11.5 Dilatación de objetos.

Este código se encuentra implementado en el archivo de proyecto 'FattenItems.m', este archivo es empleado como una función Matlab que se invoca desde el fichero principal. Para que esta función pueda funcionar recibirá como parámetros de entrada la estructura de datos que contiene tanto el vídeo como sus propiedades, estructura 'Video', la estructura de datos donde se van almacenando todos los datos obtenidos 'FrameData' y por último un parámetro que nos indica en que iteración nos encontramos.

Nada más invocar a la función la primera tarea que realizaría esta sería cargar la imagen binaria tras habérsele aplicado el filtrado de píxeles, esta imagen se extrae de la estructura global 'FrameData'. Además, en esta ocasión es importante inicializar una nueva matriz para actualizar el fotograma e ir almacenando allí los nuevos píxeles engordados en lugar de usar la propia imagen. La razón de esto es la siguiente: si sobre la propia imagen donde estamos leyendo los píxeles actualizamos estos engordándolos, en una futura iteración el algoritmo detectaría los nuevos píxeles actualizados como píxeles originales del objeto y los volvería a engordar de forma infinita, este error se representa en la figura 11-6. En su lugar lo que haremos será leer la imagen original y los píxeles engordados serán actualizados en la nueva matriz generada.

```
function [FrameData] = FattenItems(Video,FrameData,n)

I = FrameData(n-1).images.clean_image;
New_I = zeros(Video.M,Video.N);
size = 1;
```

Por último, antes de entrar en la estructura principal de la función, se debe escoger como parámetro de diseño cuanto aumentaremos la dimensión del píxel por medio de la variable 'size'. Esta actúa igual que las ventanas que se han visto en anexos anteriores de modo que si a esta variable le damos un valor de 1 el píxel pasaría de ocupar un espacio 1x1 a un espacio 3x3, de darle valor 2 pasaría a 5x5 y así sucesivamente.

Con esto se entraría en la estructura principal de lectura de imágenes píxel a píxel, el doble bucle 'for', y se comenzarían a buscar píxeles con valor de uno.

```
for ii= 1:Video.M
    for jj=1:Video.N

        if(I(ii,jj) == 1)

           lim_ii_sup = ii + size;
           lim_ii_inf = ii - size;

           if lim_ii_sup > Video.M
               lim_ii_sup = Video.M;
           end

           if lim_ii_inf < 1
               lim_ii_inf = 1;
           end

           lim_jj_sup = jj + size;
           lim_jj_inf = jj - size;

           if lim_jj_sup > Video.N
               lim_jj_sup = Video.N;
           end

           if lim_jj_inf < 1
               lim_jj_inf = 1;
           end

           New_I(lim_ii_inf : lim_ii_sup, lim_jj_inf : lim_jj_sup) = 1;


        end

    end
end
```

Como se observa esta función no goza de gran complejidad, pues únicamente genera una ventana entorno al píxel original y da un valor de 1 todas las posiciones que ocupa esta ventana en la nueva matriz. La única consideración que se debe tener en cuenta es nuevamente garantizar que las ventanas entorno a los píxeles no rebasan los límites de la imagen saturando estas, así evitamos que el algoritmo trate de dar valores a píxeles inexistentes y que Matlab deje de funcionar. Los resultados de esta función se muestran en las figuras 11-4 y 11-5.

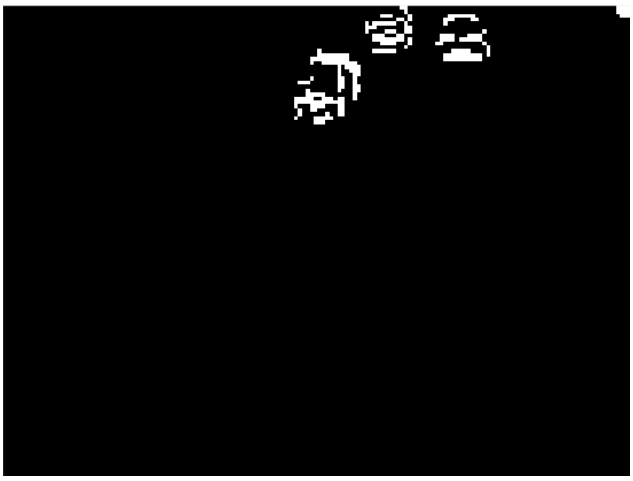

Figura 11-4. Fotograma previo a la dilatación de objetos.

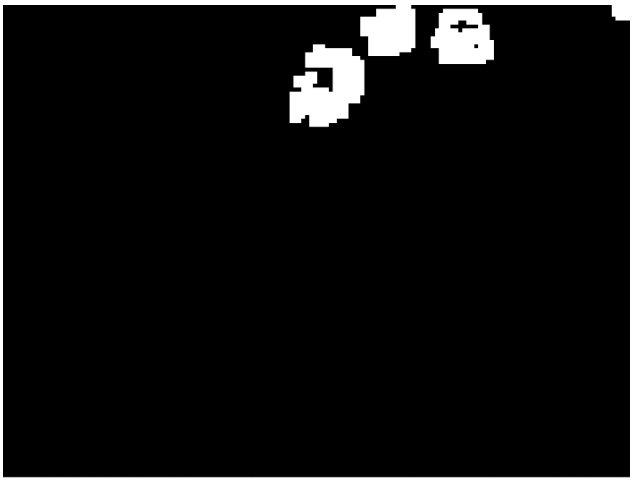

Figura 11-5. Fotograma tras la dilatación de objetos.

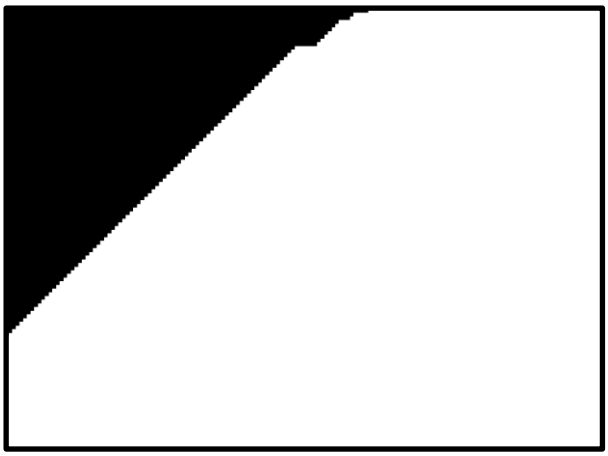

Figura 11-6. Error al actualizar estados en matriz en vez de utilizar una nueva.

Finalmente se almacena la imagen con los objetos engordados en la estructura ‘FrameData’ para que pueda ser usada en pasos posteriores.

11.6 Diferenciación de objetos.

Este código se encuentra implementado en el archivo de proyecto 'CountItems.m', este archivo es empleado como una función Matlab que se invoca desde el fichero principal. Para que esta función pueda funcionar recibirá como parámetros de entrada la estructura de datos que contiene tanto el vídeo como sus propiedades, estructura 'Video', la estructura de datos donde se van almacenando todos los datos obtenidos 'FrameData' y por último un parámetro que nos indica en que iteración nos encontramos.

La función comienza cargando el fotograma tras su último procesamiento, la dilatación de píxeles, tras esto inicializa un contador que servirá tanto para conocer la cuenta total de objetos en el fotograma actual como para ir dándole valores a los píxeles en función del objeto en el que se encuentren.

```
function [FrameData] = CountItems(Video,FrameData,n)

    I = FrameData(n-1).images.fat_items_image;
    n_object = 1;
```

Tras esto se comenzará la lectura de píxeles por medio de la estructura del doble bucle 'for' vista anteriormente en busca de píxeles que no hayan sido procesados como pertenecientes a algún objeto en particular, píxeles con valor a 1.

En el momento en que uno de estos píxeles es localizado se entraría en una nueva sección de código, en esta se realizaría un cambio de variables para referenciar las coordenadas del píxel localizado sin traer conflictos con las variables de lectura del bucle, se actualizaría el contador de objetos aumentando este y se actualizaría el valor del píxel al valor que acaba de adquirir el contador. Acto seguido, se procedería a realizar el inundado del objeto encontrado, pero para esto es necesario la inicialización de una serie de variables:

• 'V_obj_ii', vector donde se almacenan las coordenadas 'i' de los píxeles localizadas como parte del objeto durante el proceso de inundado.

• 'V_obj_jj', vector donde se almacenan las coordenadas 'j' de los píxeles localizados como parte del objeto durante el proceso de inundado.

• 'V_obj_size', contador de píxeles en los vectores anteriores.

• 'Object_size', contador de píxeles que componen el objeto procesado.

```
for ii=1:Video.M
    for jj=1:Video.N

        if I(ii,jj) == 1

            x = ii;
            y = jj;

            n_object = n_object + 1;

            obj_value = n_object;
            I(x,y) = obj_value;

            v_obj_ii = zeros();
            v_obj_jj = zeros();
            v_obj_size = 0;

            object_size = 0;
```

Declaradas estas variables se comienza el inundado, para este se entra en un bucle infinito, dado que no sabemos cuántos píxeles constituyen el objeto, de este bucle solo se saldrá cuando se den unas condiciones que se verán más adelante.

Dentro de este bucle se comienza generando la ventana que cubrirá la vecindad 3x3 entorno al píxel en el que se encuentra el algoritmo, en la creación de esta ventana se atenderá el tema de la saturación de esta para que no desborde los límites de la imagen como se ha visto en funciones anteriores. Junto a esto se inicializa una variable llamada 'change' que notificará si se ha producido algún cambio en la iteración de inundado actual.

```
change = 0;

% Saturacion ventana 3x3
% Filas
lim_inf_ii = x-1;
lim_sup_ii = x+1;

if lim_inf_ii < 1
    lim_inf_ii = 1;
end

if lim_sup_ii > Video.M
    lim_sup_ii = Video.M;
end

% Columnas
lim_inf_jj = y-1;
lim_sup_jj = y+1;

if lim_inf_jj < 1
    lim_inf_jj = 1;
end

if lim_sup_jj > Video.N
    lim_sup_jj = Video.N;
end
```

Creada la ventana se procede a la búsqueda de píxeles vecinos sin procesar dentro de la misma, es entonces cuando todo píxel con valor a 1 que se encuentre actualiza su valor al valor que tiene asignado el objeto actual, junto a esto las coordenadas de estos píxeles pasan a almacenarse en los vectores declarados anteriormente para ser analizados en iteraciones siguientes, además el flag 'change' cambia a 1 para notificar que en la iteración se ha producido una actualización y se incrementa el contador de píxeles en los registros.

```
% Actualización vecindad del pixel
for xx = lim_inf_ii : lim_sup_ii
    for yy = lim_inf_jj : lim_sup_jj

        if I(xx,yy) == 1
            % Si pixel de la vecindad vale 1 se
            % considera pixel del objeto
            change = 1;
            I(xx,yy) = obj_value;
            v_obj_size = v_obj_size + 1;
            v_obj_ii(v_obj_size) = xx;
            v_obj_jj(v_obj_size) = yy;

        end

    end
end
```

Lo siguiente que se encuentra en el código es la condición para la ruptura del bucle 'while', de este bucle solo se podrá salir cuando se den dos condiciones:

1- No se ha producido actualización en la iteración actual sobre los registros, 'change' vale 0.

2- El tamaño de los vectores de registros es igual al contador de píxeles totales en el objeto, esto notificaría que todos los píxeles almacenados en los registros ya han sido analizados.

De no cumplirse estas condiciones se actualizaría el contador de píxeles totales que conforman el objeto y se tomaría un nuevo píxel de los registros para continuar con la inundación del objeto.

```
if change == 0 && v_obj_size == object_size
    break;
end

object_size = object_size + 1;

if v_obj_size ~= object_size

    x = v_obj_ii(object_size);
    y = v_obj_jj(object_size);
    I(x,y) = obj_value;

end
```

Tarde o temprano se acabaría saliendo del bucle 'while' y se continuaría con la lectura de la imagen en busca de nuevos objetos para repetir el proceso de inundado.

Como consideración final antes de almacenar el fotograma resultado y la cuenta de objetos se realiza una pequeña corrección. Como el contador de objetos para evitar conflictos parte de un valor de 2, y por tanto todos los píxeles del primer objeto tienen este valor, se realiza un barrido donde se decrementa el valor de cada píxel de la imagen y del contador de objetos de forma que ahora el primer objeto localizado tiene todos sus píxeles a 1 y así sucesivamente con todos los objetos.

```
n_object = n_object - 1;
FrameData(n-1).n_items = n_object;

% Actualizamos tambien imagen para que el primer objeto aparezca como
% un conjunto de 1 y no de 2
for ii=1:Video.M
    for jj=1:Video.N
        I(ii,jj) = I(ii,jj) - 1;
        if I(ii,jj) <0
            I(ii,jj) = 0;
        end
    end
end

FrameData(n-1).images.count_image = I;
```

11.7 Obtención fronteras difusas.

Este código se encuentra implementado en el archivo de proyecto 'FuzzyBorders.m', este archivo es empleado como una función Matlab que se invoca desde el fichero principal. Para que esta función pueda funcionar recibirá como parámetros de entrada la estructura de datos que contiene tanto el vídeo como sus propiedades, estructura 'Video', la estructura de datos donde se van almacenando todos los datos obtenidos 'FrameData' y por último un parámetro que nos indica en que iteración nos encontramos.

La función comienza cargando el fotograma tras la dilatación de objetos e inicializando un vector de contadores con una dimensión igual al número de objetos en el fotograma, el número de objetos es un parámetro que se ha obtenido durante la diferenciación de objetos, cada componente de este vector de contadores se usará para llevar a cabo la indexación de los píxeles de las fronteras en función del objeto al que pertenezcan.

Tras esta inicialización entraremos nuevamente en la lectura píxel a píxel de la imagen por medio de la estructura de bucle 'for' en busca de píxeles con valores distintos de cero, estos son píxeles pertenecientes a objetos.

```
function [FrameData] = FuzzyBorders(Video,FrameData,n)

I = FrameData(n-1).images.count_image;

counters = zeros(1,FrameData(n-1).n_items);

for ii=1:Video.M
    for jj= 1:Video.N

        if I(ii,jj) > 0
```

Cuando se encuentre algún píxel que cumpla la condición dada se producirá lo siguiente:

1. Se tomará el valor del píxel, este nos da la referencia sobre el objeto al que pertenece el píxel y nos dirá donde almacenar los píxeles frontera asociados.

2. Se realizará un cambio de variables respecto a las coordenadas del píxel localizado para evitar conflictos con las variables de lectura de los bucles.

3. Se creará entorno a este píxel una ventana 3x3 para buscar puntos potenciales de frontera, estos son píxeles con valor de cero en la vecindad 3x3.

4. Se almacenará los píxeles en su conjunto frontera correspondiente.

```
            % value referencia al valor del objeto
            value = I(ii,jj);

            % Cambio de variables para evitar conflictos
            x = ii;
            y = jj;

            % Saturacion para ventana (3x3)
            size_window = 1;
            % Filas
            lim_inf_ii = x -size_window;
            lim_sup_ii = x + size_window;

            if lim_inf_ii < 1
                lim_inf_ii = 1;
            end
            if lim_sup_ii > Video.M
                lim_sup_ii = Video.M;
            end

            % Columnas
            lim_inf_jj = y - size_window;
            lim_sup_jj = y + size_window;

            if lim_inf_jj < 1
                lim_inf_jj = 1;
            end

            if lim_sup_jj > Video.N
                lim_sup_jj = Video.N;
            end
```

Para llevar a cabo el almacenamiento de los puntos se creará una estructura de fronteras con tantos campos como objetos contenga el fotograma, cada campo a su vez se dividirá en dos vectores, uno para guardar la fila en la que se encuentra el píxel y otro para guardar la columna. Entonces cada vez que se encuentra un píxel frontera se utilizará el identificador de objeto para:

1- Actualizar el vector de contadores, accedemos a la componente del vector con el valor de identificador de objeto pues aquí se guarda el índice actual de la frontera en la que se tiene que guardar el píxel.
2- Acceder a la estructura de datos correspondiente al objeto y junto al índice obtenido del vector de contadores se almacenará.

Como consideración final en el código se incluye una condición que dictamina que si el píxel del objeto se encuentra en algún borde de la imagen este sea considerado como píxel de frontera y no de objeto, con lo que se evita la aparición de fronteras incompletas.

```
%% Almacenado fronteras

for xx = lim_inf_ii : lim_sup_ii
    for yy = lim_inf_jj : lim_sup_jj

        if I(xx,yy) == 0
            % Actualizacion contador frontera
            counters(1,value) = counters(1,value) + 1;
            index = counters(1,value);
            % Almacen
            borders(value).ii_comp(index) = xx;
            borders(value).jj_comp(index) = yy;

        end

        % Si el pixel forma parte de borde imagen se considera
        % frontera tambien

        if xx==1 || xx==Video.M || yy==1 || yy==Video.N
            counters(1,value) = counters(1,value) + 1;
            index = counters(1,value);
            % Almacen
            borders(value).ii_comp(index) = xx;
            borders(value).jj_comp(index) = yy;
        end

    end
end
```

11.8 Procesamiento de fronteras difusas.

Este código se encuentra implementado en el archivo de proyecto 'ProcessBorders.m', este archivo es empleado como una función Matlab que se invoca desde el fichero principal. Para que esta función pueda funcionar recibirá como parámetros de entrada la estructura de datos donde se van almacenando todos los datos obtenidos 'FrameData' y por último un parámetro que nos indica en que iteración nos encontramos.

Esta función comenzará leyendo el parámetro que le indica cuantos objetos aparecen en el fotograma correspondiente a la iteración actual. Con este parámetro se entrará en un bucle 'for' que se recorrerá tantas veces como objetos en movimiento se hayan identificado.

En cada iteración de este bucle se accederá a la frontera difusa, obtenida en el anexo anterior, correspondiente al objeto de la iteración actual. Tras la extracción de esta frontera se obtendrán los valores extremos, mínimos y máximos, tanto de las filas como de las columnas, esto nos permitirá graficar las fronteras como rectángulos perfectos y conocer de una forma más eficiente entre que píxeles se encuentran los objetos en cada momento.

```
function [FrameData] = ProcessBorders(FrameData,n)

n_obj = FrameData(n-1).n_items;

for object = 1:n_obj

    comp_ii = FrameData(n-1).borders.fuzzy_borders(object).ii_comp;
    comp_jj = FrameData(n-1).borders.fuzzy_borders(object).jj_comp;

    % Se obtienen limites superiores e inferiores
    L1 = max(comp_ii);
    L2 = min(comp_ii);
    L3 = max(comp_jj);
    L4 = min(comp_jj);

    FrameData(n-1).borders.processed_borders(object).L1 = L1;
    FrameData(n-1).borders.processed_borders(object).L2 = L2;
    FrameData(n-1).borders.processed_borders(object).L3 = L3;
    FrameData(n-1).borders.processed_borders(object).L4 = L4;

end

end
```

11.9 Eliminación de objetos interiores.

Este código se encuentra implementado en el archivo de proyecto 'DeleteInnerItems.m', este archivo es empleado como una función Matlab que se invoca desde el fichero principal. Para que esta función pueda funcionar recibirá como parámetros de entrada la estructura de datos donde se van almacenando todos los datos obtenidos 'FrameData' y por último un parámetro que nos indica en que iteración nos encontramos.

La función comienza leyendo el número de objetos que se han detectado en el fotograma e inicializando dos variables que se necesitarán más adelante, por un lado, un vector donde se almacenarán en caso de que aparezcan los índices de los objetos a eliminar y una variable 'match' que notificará si se ha dado el caso de aparición de un objeto interior.

```
function FrameData = DeleteInnerItems(FrameData,n)

n_obj = FrameData(n-1).n_items;

items_to_delete = [];
match = 0;
```

Con esto se entraría en un bucle que se repetirá tantas veces como objetos tenga el fotograma. Nada más entrar a este bucle se cargarán los valores límites de la frontera correspondiente al objeto para la iteración actual. Obtenidos estos valores entraremos a un nuevo bucle dentro del anterior que también se repetirá tantas veces como objetos tenga el fotograma y cargará en cada una de sus iteraciones nuevamente los valores extremos de las fronteras de los objetos. De esta forma lo que se consigue es que cada objeto del fotograma se compare con el resto en cada iteración del primer bucle.

```
for object = 1:n_obj

    L1 = FrameData(n-1).borders.processed_borders(object).L1;
    L2 = FrameData(n-1).borders.processed_borders(object).L2;
    L3 = FrameData(n-1).borders.processed_borders(object).L3;
    L4 = FrameData(n-1).borders.processed_borders(object).L4;


    for kk = 1:n_obj

        l1 = FrameData(n-1).borders.processed_borders(kk).L1;
        l2 = FrameData(n-1).borders.processed_borders(kk).L2;
        l3 = FrameData(n-1).borders.processed_borders(kk).L3;
        l4 = FrameData(n-1).borders.processed_borders(kk).L4;

        counter = 0;
```

Teniendo cargadas dos parejas de fronteras lo primero que se comprobará es que la frontera no se está comparando consigo misma, ya que al estar cargándose las fronteras en dos bucles distintos siempre aparecerá esta por duplicado, si permitiésemos que esta se comparase consigo misma siempre se eliminaría el objeto en sí. Además, se inicializará una variable que llevará la cuenta de cuantas de las condiciones para que un objeto sea considerado interior a otro se cumplen.

Recordemos que estas condiciones hacen referencia a que los límites del objeto se encuentren contenidos en el interior de los límites del objeto con el que se comparan.

```
counter = 0;

if (~(L1==l1 && L2==l2 && L3==l3 && L4==l4))

    cond1 = (L1 >= l1) && (l1 >= L2);
    if cond1
        counter = counter +1;
    end

    cond2 = (L2 <= l2) && (l2 <= L1);
    if cond2
        counter = counter + 1;
    end

    cond3 = (L3 >= l3) && (l3 >= L4);
    if cond3
        counter = counter + 1;
    end

    cond4 = (L4 <= l4) && (l4 <= L3);
    if cond4
        counter = counter + 1;
    end

    if counter > 2
        items_to_delete = [items_to_delete kk];

        match = 1;
    end

end
```

Cuando se cumplen más de dos condiciones se considerará que un objeto es interior a otro y se almacenará su índice en el vector de objetos a borrar. Finalizado el doble bucle en este vector se incluirán los índices de todos los objetos que se deben borrar.

Se comienza entonces un bucle con tantas iteraciones como objetos a borrar se tengan, este es importante leerlo de forma descendente, si no se hace así al eliminar objetos estaríamos cambiando la dimensión del vector y en el siguiente acceso el índice al que se accede no sería el indicado y se acabaría produciendo un error en el caso de tratar de acceder a un índice inexistente en el vector a causa de las redimensiones de este.

Importante aclarar que, cuando se habla de borrar un objeto, lo que se borra es su frontera de forma que esta no se visualiza en la secuencia de vídeo y no se tiene en cuenta en funciones siguientes.

Al final de la función se actualiza el contador de objetos total del fotograma retirando de este los objetos eliminados.

```
if match == 1

   [~,nn] = size(items_to_delete);


   %rem = sort(rem,'descend');
   for ii = nn:1
       FrameData(n-1).borders.processed_borders(items_to_delete(ii)) = [];
   end

   FrameData(n-1).n_items = FrameData(n-1).n_items - nn;

end
```

11.10 Obtención de puntos singulares.

Este código se encuentra implementado en el archivo de proyecto 'SingularPoints.m', este archivo es empleado como una función Matlab que se invoca desde el fichero principal. Para que esta función pueda funcionar recibirá como parámetros de entrada la estructura de datos que contiene tanto el vídeo como sus propiedades, estructura 'Video', la estructura de datos donde se van almacenando todos los datos obtenidos 'FrameData' y por último un parámetro que nos indica en que iteración nos encontramos.

La función comenzará cargando el fotograma con los objetos en movimiento detectados, el parámetro que nos indica cuantos objetos en movimiento hay en el fotograma actual e inicializará un vector vacío para el almacenamiento de los puntos singulares que se detecten en el fotograma.

```
function FrameData = SingularPoints(Video,FrameData,n)

I = FrameData(n-1).images.borders;
n_obj = FrameData(n-1).n_items;
PS_frame = [];
```

Tras esto se entrará en la estructura principal de la función, esta es un bucle del tipo 'for' que tendrá tantas iteraciones como objetos en movimiento se hayan detectado. Entonces para cada objeto se extraerán los valores límite de sus fronteras, los cuales se habían almacenado en la estructura de datos global 'FrameData', y dentro de estos se realizará la búsqueda de puntos singulares por medio de la función de Matlab 'corner', se le indicará a esta como opción el empleo del método 'Harris'. Además, como parámetro de la función 'corner' se da el número máximo de puntos que queremos que esta devuelva, aunque debido al bajo número de píxeles que conforman nuestros objetos este máximo nunca se alcanza.

```
for object = 1:n_obj

        L1 = FrameData(n-1).borders.processed_borders(object).L1;
        L2 = FrameData(n-1).borders.processed_borders(object).L2;
        L3 = FrameData(n-1).borders.processed_borders(object).L3;
        L4 = FrameData(n-1).borders.processed_borders(object).L4;

        PS_object = corner(I(L2:L1,L4:L3),'Harris',50);
```

Lo siguiente que se hará será comprobar si para el objeto analizado se ha obtenido al menos un punto singular pues se observó que, sobre todo, al inicio de la secuencia de vídeo cuando el objeto es especialmente pequeño podía darse que en su interior no se detectase ningún punto. Como es imperativo que los objetos tengan al menos un punto singular para poder realizar el futuro emparejamiento entre fotogramas, para estos casos se generará manualmente un punto, que, aunque no sea singular se considerará como tal. Este punto generado manualmente será el centro geométrico del objeto.

Además, como la función 'corner' había realizado la búsqueda de puntos singulares en el interior de los límites del objeto las coordenadas de estos puntos obtenidos se encuentran en el marco de referencia de los límites del objeto y no de la imagen, por lo que se realiza una transformación de sus coordenadas para que vuelvan al marco de referencia de la imagen.

```
        if(isempty(PS_object))
            PS_object(1,2) = fix((L1+L2)/2);
            PS_object(1,1) = fix((L3+L4)/2);
        else
            PS_object(:,1) = PS_object(:,1) + L4 - 1;
            PS_object(:,2) = PS_object(:,2) + L2 - 1;
        end
```

Obtenidos los puntos singulares del objeto asociaremos a cada uno de estos cierta información que facilite su

emparejamiento, esta información se obtiene a partir del cálculo del tensor 'T'. Para esto se construirá un nueva función llamada 'TensorCalculus.m' que recibirá como parámetros de entrada las estructuras de datos 'Vídeo' y 'FrameData', el conjunto de puntos singulares obtenido y los parámetros que indican en que iteración nos encontramos y para que objeto estamos calculando los tensores.

```
FrameData = TensorCalculus(Video,FrameData,PS_object,n,object);
```

Esta nueva función comenzará comprobando cuantos puntos singulares se han detectado, cargará el fotograma actual en escala de grises y cambiará su formato de dato al tipo 'double', es sobre esta imagen sobre la que se aplica el cálculo del tensor 'T'.

Entonces sobre esta imagen se realiza el cálculo aproximado de gradiente de la imagen tanto en la componente 'x' como en la componente 'y', valores necesarios para el cálculo de 'T'.

```
function FrameData = TensorCalculus(Video,FrameData,PS_object,n,object)

[n_points,~] = size(PS_object);

I = FrameData(n-1).images.I1_gray;
Id = double(I);

Ix = zeros(Video.M,Video.N);
Iy = zeros(Video.M,Video.N);
Ix(:,2:Video.N) = Id(:,2:Video.N) - Id(:,1:Video.N-1);
Iy(2:Video.M,:) = Id(2:Video.M,:) - Id(1:Video.M-1,:);
```

Con todos estos parámetros obtenidos se pasaría a la estructura principal de la función, esta es una estructura de bucle 'for' que se repite tantas veces como puntos singulares se hayan encontrado.

En cada iteración de este bucle se inicializará un nuevo tensor y el tamaño de la vecindad sobre la que lo calcularemos por medio de la variable 'S', esta se ha fijado a 4 lo que significa que calcularemos el tensor para una vecindad de 9x9. Entonces para cada píxel de la vecindad, considerando como en apartados anteriores que la vecindad no desborda la imagen, se obtendrá su valor del tensor a partir de los gradientes previamente obtenidos y se acumularán en el interior del tensor total, ya que este se está acumulando como un sumatorio de los tensores de cada píxel de la vecindad.

Por último, una vez calculado el tensor para el píxel se almacenará la información relevante de la vecindad de este en un vector, con información se hace referencia a cada una de las componentes del tensor las cuales se normalizarán entre cero y uno para guardarse en la estructura global de datos 'FrameData'.

```
for tt = 1:n_points

T = zeros(2,2);
S = 4;

        for mm = PS_object(tt,2)-S : PS_object(tt,2)+S
            for nn = PS_object(tt,1)-S : PS_object(tt,1)+S

                ii = mm;
                jj = nn;

                if ii<1
                    ii=1;
                end

                if jj<1
                    jj=1;
                end

                if ii>Video.M
                    ii=Video.M;
                end

                if jj>Video.N
                    jj=Video.N;
                end

                T = T + [ Ix(ii,jj)^2, Ix(ii,jj)*Iy(ii,jj); ...
                    Ix(ii,jj)*Iy(ii,jj), Iy(ii,jj)^2];

            end
        end

        u = [T(1,1),T(1,2),T(2,2)];
        c_info = u/norm(u);

        PS_object(tt,3) = c_info(1);
        PS_object(tt,4) = c_info(2);
        PS_object(tt,5) = c_info(3);
end

FrameData(n-1).object_info(object).s_points = PS_object;
```

11.11 Emparejamiento de puntos entre fotogramas.

Este código se encuentra implementado en el archivo de proyecto 'pointsPairs.m', este archivo es empleado como una función Matlab que se invoca desde el fichero principal. Para que esta función pueda funcionar recibirá como parámetros de entrada la estructura de datos donde se van almacenando todos los datos obtenidos 'FrameData', un parámetro que nos indica en que iteración nos encontramos 'n' y un contador donde se almacena cuantos objetos diferentes se han identificado hasta el momento 'Global_identifier'.

Al comenzar la función se partirá extrayendo de la estructura de datos 'FrameData' los parámetros que determinan cuantos objetos han sido detectados en dos fotogramas consecutivos. Con estos datos se inicializarán dos variables:

- 'n_objects': Objetos detectados en la iteración actual.
- 'n_objects_old': Objetos detectados en la iteración anterior.

Tras esto se realizará una comprobación para ver si nos encontramos en la primera ocasión en la que se entra en esta función, ya que hasta entonces no habría sido identificado ningún objeto.

La primera vez que se accede a esta función es en la tercera iteración donde ya se han leído tres fotogramas y por tanto se han realizado todos los procesos para detectar el movimiento en dos parejas de estos. En esta primera iteración nos encontramos con que los objetos en movimiento que se han detectado en el fotograma no pueden ser identificados al no existir un fotograma anterior para comparar. Por tanto, para este primer caso la identificación se realiza de manera manual.

Se leerán en un bucle 'for' todos los objetos en movimiento detectados y se les asignará un identificador único, a modo de contador incremental, y a su vez se actualizará la variable 'Global_identifier' con el último identificador asignado de forma que cuando aparezca más adelante un nuevo objeto se le pueda asignar un nuevo identificador siguiendo la secuencia incremental.

```
function [FrameData,Global_identifier] = pointsPairs(FrameData,n,Global_identifier)


n_objects = FrameData(n-1).n_items;
n_objects_old = FrameData(n-2).n_items;

if n == 3
    for zz=1:n_objects_old
        FrameData(n-2).object_info(zz).identifier = zz;
        Global_identifier = Global_identifier + 1;
    end
end
```

Tras esta consideración para el caso inicial se entraría en la estructura principal de la función, esta se compone por un total de cuatro bucles 'for' anidados donde en cada uno se recorren las siguientes variables:

1. Número de objetos a identificar en el fotograma actual.
2. Número de objetos identificados en el fotograma anterior.
3. Número de puntos singulares en cada objeto sin identificar.
4. Número de puntos singulares en cada objeto identificado.

En resumen, para cada objeto sin identificar se realizará una comparación con cada uno de los objetos identificados en el fotograma anterior. Para esto se cargarán en cada iteración los puntos singulares del objeto sin identificar y los puntos de los objetos identificados con los que se compararán.

Se comienza entonces en el primero de los bucles cargando los puntos correspondientes al objeto sin identificar que será analizado en la iteración. Además, se inicializa un vector 'object_score' con un tamaño igual a cuantos objetos han sido identificados hasta el momento, en este se irán almacenando las puntuaciones que cada objeto identificado irá acumulando como potencial pareja para el que se está tratando de identificar. Cada entrada de este vector irá asociado a un objeto de forma que el índice que devuelva la mayor puntuación se reconocerá como el ganador y el objeto a identificar heredará su identificador.

```
for kk = 1:n_objects

    points_object = FrameData(n-1).object_info(kk).s_points;
    [NP_obj,~] = size(points_object);

    object_score = zeros(1,Global_identifier);
```

Siguiendo a esta inicialización se entraría en el siguiente bucle anidado, en este nuevo bucle se irán cargando en cada iteración los puntos singulares de cada objeto identificado con los que se compararán los puntos singulares del objeto a identificar, se toma además el identificador del objeto que se está comparando en cada iteración.

```
for gg = 1:n_objects_old
     points_object_old = FrameData(n-2).object_info(gg).s_points;
    [NP_obj_old,~] = size(points_object_old);


    id = FrameData(n-2).object_info(gg).identifier;
```

En el tercero de los bucles anidados se cargarán los puntos singulares del objeto no identificado para poder ir comparándolos con los puntos singulares de cada objeto identificado en el cuarto bucle anidado.

Es dentro de este último bucle donde se realiza el cálculo para ver si dos puntos pueden considerarse como potencialmente conjugados. Para esto se cargará la información asociada a cada punto que se está comparando, recordar que esta se había obtenido en apartados anteriores por medio del cálculo del tensor en la vecindad, y se realizará el producto lo que nos devolverá una puntuación.

Con esta puntuación se comprobará que el punto sobrepase un valor umbral escogido como parámetro de diseño, en este caso fijado a 0.75, ya que para que dos puntos puedan ser tratados como parejos el resultado de su producto debe ser lo mayor posible, siendo el máximo un valor de 1 por haber aplicado la normalización. Los puntos que sobrepasen este umbral serán almacenados en un vector previamente inicializado como puntos potencialmente conjugados y además para cada uno de estos se calculará la distancia que los separa del punto con el que se comparan.

```
for ii = 1:NP_obj

    point = points_object(ii,:);
    points = [];
    dists = [];

       for jj = 1:NP_obj_old

           point_old = points_object_old(jj,:);

           score = point(3:5)*point_old(3:5)';
           dist = sqrt((point(1)-point_old(1))^2 + (point(2)-point_old(2))^2);

           if score > 0.75
              points = [points; point_old];
              dists = [dists; dist];
           end

       end
```

Habiendo comparado todos los puntos en el fotograma antiguo con el que está siendo analizado saldremos del

cuarto bucle anidado. A la salida de este veremos si se han encontrado puntos que sobrepasasen la condición anterior y en el caso afirmativo se extraerá únicamente el más próximo al que se está analizando, quedando el resto de los puntos descartados.

Este punto extraído como el conjugado pasará una condición final de mínima distancia, si la distancia que separa a los dos puntos sobrepasa un determinado valor fijado como parámetro de diseño se interpretará que finalmente el punto no era conjugado y será descartado también. Si por el contrario este si cumple la condición se comprobará a que objeto pertenece y se le sumará un punto al índice correspondiente en el vector de puntuaciones que había sido inicializado con anterioridad.

```
if ~isempty(dists)

    [minDist,index] = min(dists);

    if minDist < 6

     points(index,:);
     object_score(id) = object_score(id) + 1;
    end
end
```

Finalmente, comparados todos los puntos singulares de todos los objetos móviles en el fotograma anterior con los puntos singulares de un objeto sin identificar en el fotograma anterior se llevará a cabo la selección del identificador.

Para esto se extraerá el índice de la componente del vector de puntuaciones con mayor valor, se tomará entonces el identificador de este objeto que ha ganado en número de emparejamientos de puntos y se le dará su identificador al nuevo objeto. Aquí además puede darse el caso de que ningún objeto haya logrado emparejar ninguno de sus puntos, esto se interpretará como que el objeto es nuevo en la secuencia de vídeo y por tanto no se puede emparejar con otro objeto.

Este nuevo objeto será etiquetado haciendo uso del contador ‘Global_identifier’ en el cual se había almacenado el último identificador usado.

```
[maxVal,index] = max(object_score);

if maxVal ~= 0

    for hh= 1:n_objects_old
        if FrameData(n-2).object_info(hh).identifier == index
            objective = hh;
            break;
        end
    end

    FrameData(n-1).object_info(kk).identifier = FrameData(n-2).object_info(objective).identifier;
    FrameData(n-1).object_info(kk).frame_object = kk;
else
    Global_identifier = Global_identifier + 1;
    FrameData(n-1).object_info(kk).identifier = Global_identifier;
    FrameData(n-1).object_info(kk).frame_object = kk;
end
```

11.12 Corrección de identificadores.

Este código se encuentra implementado en el archivo de proyecto ‘checkErrors.m’, este archivo es empleado como una función Matlab que se invoca desde el fichero principal. Para que esta función pueda funcionar recibirá como parámetros de entrada la estructura de datos donde se van almacenando todos los datos obtenidos ‘FrameData’, un parámetro que nos indica en que iteración nos encontramos ‘n’ y un contador donde se almacena cuantos objetos diferentes se han identificado hasta el momento ‘Global_identifier’.

Esta función debe corregir automáticamente resultados obtenidos tras la identificación de los objetos en movimiento de forma que no aparezcan identificadores duplicados.

La función comienza cargando el parámetro que le indica cuantos objetos contiene el fotograma actual. Tras esto inmediatamente entrará en una estructura de doble bucle ‘for’ donde lo que hará será comparar los objetos que aparecen en el fotograma entre sí. En cada iteración se irán cargando una pareja de objetos y se comprobará que no tengan el mismo identificador.

En el caso que dos objetos estuviesen compartiendo un identificador el identificador del último sería actualizado haciendo uso de la variable ‘Global_identifier’ donde se almacena el siguiente identificador libre disponible.

```
function [FrameData,Global_identifier] = CheckErrors(FrameData,n,Global_identifier)

n_obj = FrameData(n-1).n_items;

for object = 1:n_obj


    object_ref = FrameData(n-1).object_info(object).frame_object;

    for object2= 1:n_obj

        object_ref_2 = FrameData(n-1).object_info(object2).frame_object;

        if object_ref ~= object_ref_2
           id1 = FrameData(n-1).object_info(object).identifier;
           id2 = FrameData(n-1).object_info(object2).identifier;

           if id1 == id2
               Global_identifier = Global_identifier + 1;
               FrameData(n-1).object_info(object).identifier = Global_identifier;
           end

        end

    end

end

end
```

11.13 Construcción fotogramas resultado.

Este código se encuentra implementado en el archivo de proyecto 'PlotBordersId.m', este archivo es empleado como una función Matlab que se invoca desde el fichero principal. Para que esta función pueda funcionar recibirá como parámetros de entrada la estructura de datos donde se van almacenando todos los datos obtenidos 'FrameData' y un parámetro que nos indica en que iteración nos encontramos 'n'.

Esta función se encarga de generar el fotograma resultado para la aplicación, el fotograma resultado deberá encuadrar los objetos en movimiento detectados por el algoritmo y acompañar a estos con un identificador.

La función comienza cargando el fotograma original que se va a modificar junto al parámetro que le indica cuantos objetos aparecen en el fotograma, además se comienza la inicialización de variables de importancia para el proceso:

• Listado de colores, se usarán para determinar el color de cada frontera.

• 'colors', objeto donde se almacena el listado de colores de las etiquetas de texto que se mostrarán en el fotograma.

• 'positions', vector donde se almacenan las coordenadas en las que se mostrará cada etiqueta de identificación.

• 'text_str', objeto donde se almacenan los textos que se mostrarán en las etiquetas.

```
function  [FrameData] = PlotBordersID(FrameData,n)

I = FrameData(n-1).images.I1_RGB;
n_obj = FrameData(n-1).n_items;


red = [255,0,0];
green = [0,255,0];
yellow = [255,255,0];
blue = [0,0,255];
black = [0,0,0];
cyan = [0,255,255];
magenta = [255,0,255];


colors = {};
positions = [];
text_str = cell(1,n_obj);
```

Tras estas inicializaciones se entrará en un bucle 'for' con tantas iteraciones como objetos aparezcan en el fotograma.

Una vez dentro de este bucle se cargará el identificador del objeto que está siendo procesado en la iteración actual. Un problema que se encontró en el desarrollo de esta función, al perseguirse que cada objeto enmarcado apareciese con un color propio distinto a los demás, fue que la función de Matlab 'insertText', que es la encargada de introducir las etiquetas deseadas en el fotograma, solo funcionaba para un listado de colores predefinido. Este es un listado de únicamente unos siete colores lo que impedía que para la secuencia de vídeo estos fuesen totalmente únicos para cada objeto. Lo que se implementó entonces fue que esta secuencia de colores se repitiese cada vez que se recorriese el listado de colores entero de forma que, por ejemplo, el objeto 1 y el objeto 8 compartiría el mismo. El sistema de asignación de colores a cada objeto que se construyó entonces quedó de la siguiente forma:

```
identifier = FrameData(n-1).object_info(object).identifier;

hh = fix(identifier/7);

if hh>0
    id = identifier - 7*hh+1;
else
    id = identifier;
end


if id == 1
    color_border = red;
    colors(object) = {'red'};
elseif id == 2
    color_border = green;
    colors(object) = {'green'};
elseif id == 3
    color_border = yellow;
    colors(object) = {'yellow'};
elseif id == 4
    color_border = black;
    colors(object) = {'black'};
elseif id == 5
    color_border = blue;
    colors(object) = {'blue'};
elseif id == 6
    color_border = cyan;
    colors(object) = {'cyan'};
elseif id == 7
    color_border = magenta;
    colors(object) = {'magenta'};
end
```

Lo que se puede ver en el código anterior es una conversión del identificador del vehículo de forma que este nunca exceda el máximo de colores existentes, esto identificador secundario se usará exclusivamente para la asignación de colores ya que el objeto debe conservar su identificador principal.

Lo siguiente que realiza el algoritmo es el trazado de las fronteras de los objetos detectados sobre el fotograma, para esto se extraerán las fronteras del objeto de la estructura 'FrameData' y se modificarán las coordenadas del fotograma donde aparecen asignándoles su color correspondiente.

```
 L1 = FrameData(n-1).borders.processed_borders(object).L1;
 L2 = FrameData(n-1).borders.processed_borders(object).L2;
 L3 = FrameData(n-1).borders.processed_borders(object).L3;
 L4 = FrameData(n-1).borders.processed_borders(object).L4;

I(L2,L4:L3,1) = color_border(1);
I(L2,L4:L3,2) = color_border(2);
I(L2,L4:L3,3) = color_border(3);

I(L1,L4:L3,1) = color_border(1);
I(L1,L4:L3,2) = color_border(2);
I(L1,L4:L3,3) = color_border(3);

I(L2:L1,L3,1) = color_border(1);
I(L2:L1,L3,2) = color_border(2);
I(L2:L1,L3,3) = color_border(3);

I(L2:L1,L4,1) = color_border(1);
I(L2:L1,L4,2) = color_border(2);
I(L2:L1,L4,3) = color_border(3);
```

Por último, se generaría la etiqueta asociada al vehículo. Para esto únicamente se debe dar el texto que se quiere visualizar, en este caso el identificador del vehículo, y la posición donde se quiere ver la etiqueta, se ha escogido la esquina inferior izquierda de la frontera. En cada iteración esta información creada se almacena en los vectores inicializados en la primera parte de la función.

```
    text_str{object} = num2str(identifier);
    positions = [positions; (L4-15) L1];
```

Tras haber recorrido todos los objetos en el bucle se generaría el fotograma resultado introduciendo las etiquetas construidas por medio de la función Matlab 'insertText'. Se incluye una condición final que se encarga de que en el caso de que no haya objetos detectados en el fotograma no se invoque esta función ya que de invocarse sin haber generado los parámetros necesarios provocaría un error para el programa. Finaliza entonces con el almacenamiento del fotograma en la estructura 'FrameData' y mostrando este por pantalla para el usuario.

```
if isempty(positions)
    RGB = I;
else
    RGB = insertText(I,positions,text_str,'FontSize',10,'BoxColor',colors);
end
FrameData(n-1).images.color_borders = RGB;

figure(500);
hold on
title(n)
imshow(RGB,'InitialMagnification',400);
```

11.14 Generación de centros.

Este código se encuentra implementado en el archivo de proyecto 'GenerateCenters.m', este archivo es empleado como una función Matlab que se invoca desde el fichero principal. Para que esta función pueda funcionar recibirá como parámetros de entrada la estructura de datos donde se van almacenando todos los datos obtenidos 'FrameData' y un parámetro que nos indica en que iteración nos encontramos 'n'.

Esta función comienza cargando un parámetro de la estructura 'FrameData' que le indica la cantidad de objetos en movimiento que han sido detectados en el fotograma.

Tras esto entra en un bucle 'for' con tantas iteraciones como objetos detectados, donde se extraerán los límites calculados del objeto y a partir de estos se obtendrá el punto central geométrico como la intersección entre la mitad de su base y de su altura.

```
function [FrameData] = GenerateCenters(FrameData,n)

    n_obj = FrameData(n-1).n_items;

    for object = 1:n_obj

        L1 = FrameData(n-1).borders.processed_borders(object).L1;
        L2 = FrameData(n-1).borders.processed_borders(object).L2;
        L3 = FrameData(n-1).borders.processed_borders(object).L3;
        L4 = FrameData(n-1).borders.processed_borders(object).L4;

        FrameData(n-1).object_info(object).center(1,2) = fix((L1+L2)/2);
        FrameData(n-1).object_info(object).center(1,1) = fix((L3+L4)/2);


    end

end
```

11.15 Obtención de las trayectorias.

Este código se encuentra implementado en el archivo de proyecto 'ItemPath.m', este archivo es empleado como una función Matlab que se invoca desde otra función llamada 'PlotBordersId.m'. Para que esta función pueda operar debe recibir como parámetros de entrada los siguientes argumentos:

- 'FrameData': Estructura de datos donde se recopilan todos los datos del algoritmo.
- 'n': Iteración del algoritmo en la que se encuentra.
- 'object': Identificador de objeto a analizar en el marco del fotograma.
- 'identifier': Identificador de objeto a analizar en el marco de la secuencia completa.
- 'trayectorias': Estructura de datos donde se van almacenando las trayectorias calculadas.

Esta función es invocada en el interior de la función 'PlotBordersId.m' en el momento que van a ser graficados las fronteras de los objetos detectados. Entonces para cada uno de los objetos se entrará en esta nueva función.

Acceder a la función implica que se está analizando un objeto, entonces se comienza extrayendo el centro calculado de este objeto. Tras la extracción del centro se buscará la existencia de este mismo objeto en un fotograma anterior.

Para realizar la búsqueda se construye un bucle 'for' que leerá todos los objetos del fotograma anterior en busca del identificador del objeto que está siendo analizado. En caso de coincidencia se producirá una bifurcación, ya que el objeto antiguo puede tener ya una trayectoria vinculada o no. En caso afirmativo se entrará en un proceso de actualización de trayectoria, función 'continuePath', o de no existir esta trayectoria se comenzará a crear una nueva, función 'newPath'.

```
function [FrameData,trayectorias] = ItemPath(FrameData,n,object,identifier,trayectorias)

% Cogemos centro del objeto
 c_ii = FrameData(n-1).object_info(object).center(2);
 c_jj = FrameData(n-1).object_info(object).center(1);
 centro = [c_ii; c_jj];

 % Buscamos el centro del mismo objeto en el frame anterior
 objects_old = FrameData(n-2).n_items;

 for kk = 1:objects_old
     % Si encuentra coincidencia
    if FrameData(n-2).object_info(kk).identifier == identifier
        % Miramos si ya tiene trayectoria creada
        if isfield(FrameData(n-2).object_info(kk),'trayectoria') &&(~isempty(FrameData(n-2).object_info(kk).trayectoria))
            trayectoria = FrameData(n-2).object_info(kk).trayectoria;
            trayectoria = continuePath(trayectoria,centro);

            FrameData(n-1).object_info(object).trayectoria = trayectoria;
            trayectorias.items(identifier).path = trayectoria;
        else
            c_ii_old = FrameData(n-2).object_info(kk).center(2);
            c_jj_old = FrameData(n-2).object_info(kk).center(1);
            centro_old = [c_ii_old; c_jj_old];
            trayectoria_update = newPath(centro,centro_old);

            FrameData(n-1).object_info(object).trayectoria = trayectoria_update;
            trayectorias.items(identifier).path = trayectoria_update;
        end

    end


 end


end
```

- **Función 'newPath'.**

Esta función debe recibir los centros del objeto en los dos fotogramas consecutivos para poder trazar los píxeles que los unen.

La mayor complejidad para el desarrollo de esta función reside en la consideración de todos los casos a la hora del trazado de los píxeles de unión.

Se comienza calculando las diferencias entre los centros tanto en número de filas como de columnas.

```
dif_ii = centro(1) - centro_old(1);
dif_jj = centro(2) - centro_old(2);

dif_ii = abs(dif_ii);
dif_jj = abs(dif_jj);
```

Teniendo estas diferencias se comenzarán a abordar los diferentes casos posibles:

- Misma diferencia de filas y de columnas.
- Mayor diferencia de filas que de columnas.
- Mayor diferencia de columnas que de filas.

El caso de misma diferencia resulta bastante sencillo pues solo tiene dos subcasos, si esta diferencia es menor o igual a la unidad los propios centros conformarían la trayectoria directamente o en caso contrario se tendría que generar una diagonal con píxeles con tantos puntos como indique la diferencia.

```
if dif_ii == dif_jj

    % Centro no se ha movido
    if dif_ii <=1
        trayectoria_update = [centro_old,centro];

    else
        % Misma diferencia, mismos puntos
        N_points = dif_ii-1+2;

        % Componente ii
        tr_ii = linspace(centro_old(1),centro(1),N_points);


        % Componente jj
        tr_jj = linspace(centro_old(2),centro(2),N_points);

        trayectoria_update = [tr_ii; tr_jj];
    end
```

El siguiente caso, mayor diferencia en las filas que de columnas, ya resulta más complejo. Se comienza comprobando si la diferencia de filas es igual a la unidad, menor caso posible, esto implicaría que la diferencia de columnas es cero y los centros conformarían directamente la trayectoria. Si la diferencia de filas fuese mayor que la unidad entonces se comenzaría obteniendo la componente fila de los nuevos píxeles a añadir para la trayectoria como una línea que comienza en la coordenada fila del primer centro y termina en la coordenada fila del segundo centro con tantos puntos intermedios como marque la diferencia.

- Ejemplo demostrativo trazado de trayectoria, componente fila:

Coordenada fila centro 1: 23

Coordenada fila centro 2: 28

Coordenadas fila trayectoria resultado: [23 24 25 26 27 28]

Quedaría por calcular las componentes columna de estos píxeles. Nuevamente nos encontramos con una serie de casos que tratar. Si la diferencia de columnas fuese cero, significaría que la trayectoria es una línea recta que solo avanza en la coordenada de las filas, por lo que se rellenarían estas coordenadas creando un vector con tantos puntos como marque la trayectoria de coordenadas fila y valiendo cada componente lo mismo que la coordenada columna del centro original.

- Ejemplo demostrativo trazado de trayectoria, componente columna caso 1:

  Coordenada columna centro 1: 33

  Coordenada columna centro 2: 33

  Coordenadas columna trayectoria resultado: [33 33 33 33 33 33]

Si esta diferencia de columna valiese uno en vez de cero, significaría que solo se ha desplazado una columna, por lo que se tomarán la mitad de los puntos para las coordenadas columnas con el valor del centro original y la otra mitad con el valor del centro nuevo.

- Ejemplo demostrativo trazado de trayectoria, componente columna caso 2:

  Coordenada columna centro 1: 33

  Coordenada columna centro 2: 34

  Coordenadas columna trayectoria resultado: [33 33 33 34 34 34]

Por último, si esta diferencia fuese mayor se tomaría el número de puntos de la trayectoria de coordenadas fila y se dividiría entre la diferencia de columnas, esto daría el número de segmentos que se van a concatenar para formar la trayectoria sabiendo los valores de las coordenadas intermedias para las columnas.

- Ejemplo demostrativo:

  Coordenada columna centro 1: 33

  Coordenada columna centro 2: 35

  Coordenadas columna trayectoria resultado: [33 33 34 34 35 35]

Adicionalmente, se incluyen en el código las consideraciones para evitar problemas con los redondeos, como los redondeos se hacen siempre a la baja se añadirán los puntos que falten en el segmento final.

```
elseif dif_ii > dif_jj

    if dif_ii == 1
        trayectoria_update = [centro_old,centro];
    else

        N_points = dif_ii -1 +2;

        tr_ii = linspace(centro_old(1),centro(1),N_points);

        if dif_jj ==0
            tr_jj = zeros(1,size(tr_ii,2))+centro(2);
        elseif dif_jj == 1

            tr_jj = zeros(1,size(tr_ii,2));
            tr_jj(1:fix(N_points/2)) = centro_old(2);
            tr_jj(fix(N_points/2)+1:end) = centro(2);

        else
            tr_jj = zeros(1,size(tr_ii,2));
            segments = dif_jj-1+2;

            % Puntos Por Segmento
            ppsegment = fix(N_points/segments);

            % Puntos
            N_points_jj = dif_jj-1+2;
            points = linspace(centro_old(2),centro(2),N_points_jj);

            tr_jj = [];
            for segment = 1:segments
                tr_segment = zeros(1,ppsegment)+points(segment);
                tr_jj = [tr_jj, tr_segment];
            end

            % Distintas dimensiones, producidas por el redondeo
            if size(tr_ii,2) > size(tr_jj,2)
                tr_extra = zeros(1,size(tr_ii,2)-size(tr_jj,2))+points(end);
                tr_jj = [tr_jj, tr_extra];
            end
        end
       trayectoria_update = [tr_ii;tr_jj];
    end
```

En el tercer caso, mayor diferencia en columnas que en filas se desarrollará el mismo código, pero invirtiendo los casos entre filas y columnas.

```
elseif dif_ii < dif_jj

    if dif_jj == 1
        trayectoria_update = [centro_old,centro];
    else

        N_points = dif_jj -1 +2;
        tr_jj = linspace(centro_old(2),centro(2),N_points);

        if dif_ii == 0
            tr_ii = zeros(1,size(tr_jj,2))+centro(1);
        elseif dif_ii == 1
            tr_ii = zeros(1,size(tr_jj,2));
            tr_ii(1:fix(N_points/2)) = centro_old(1);
            tr_ii(fix(N_points/2)+1:end) = centro(1);
        else
            tr_ii = zeros(1,size(tr_jj,2));
            segments = dif_ii-1+2;
            % Puntos Por Segmento
            ppsegment = fix(N_points/segments);
            % Puntos
            N_points_ii = dif_ii-1+2;

            points = linspace(centro_old(1),centro(1),N_points_ii);


            tr_ii = [];
            for segment = 1:segments
                tr_segment = zeros(1,ppsegment)+points(segment);
                tr_ii = [tr_ii, tr_segment];
            end

            % Distntas dimensiones, producidas por el redondeo
            if size(tr_jj,2) > size(tr_ii,2)
                tr_extra = zeros(1,size(tr_jj,2)-size(tr_ii,2))+points(end);
                tr_ii = [tr_ii, tr_extra];
            end

        end
        trayectoria_update = [tr_ii;tr_jj];
    end
```

Este proceso devolvería una trayectoria que sería almacenada en el objeto analizado para una futura iteración.

- **Función ‘continuePath’.**

Esta función recibe como parámetros el centro del objeto analizado y la trayectoria que lleva recorrida hasta el fotograma anterior. Dentro de esta función se extraerá el último punto de la trayectoria recogida, lo que correspondería al centro del objeto en el fotograma anterior, y se reciclaría la función ‘newPath’ que con los dos centros nos devolvería un nuevo fragmento de la trayectoria para que lo concatenemos a lo anterior.

```
function trayectoria = continuePath(trayectoria,centro)

    centro_old = trayectoria(:,end);
    trayectoria(:,end) = [];
    trayectoria_update = newPath(centro,centro_old);
    trayectoria = [trayectoria, trayectoria_update];

end
```

Finalizada la obtención de la trayectoria para el objeto se retornaría a la función 'PlotBordersId.m' para su representación sobre el fotograma resultado, esto se implementa como se muestra en el siguiente fragmento de código.

```
[FrameData,trayectorias] = ItemPath(FrameData,n,object,identifier,trayectorias);

if isfield(FrameData(n-1).object_info(object),'trayectoria')
    trayectoria = FrameData(n-1).object_info(object).trayectoria;
    points_path = size(trayectoria,2);

    for point = 1:points_path
       I(trayectoria(1,point),trayectoria(2,point),1) = color_border(1);
       I(trayectoria(1,point),trayectoria(2,point),2) = color_border(2);
       I(trayectoria(1,point),trayectoria(2,point),3) = color_border(3);
    end
end
```

11.16 Comparativa trayectorias obtenidas por algoritmo frente a ojo humano.

En este anexo se incluyen las comparativas de las trayectorias y los respectivos errores asociados para los 10 vehículos que recorren la secuencia de vídeo.

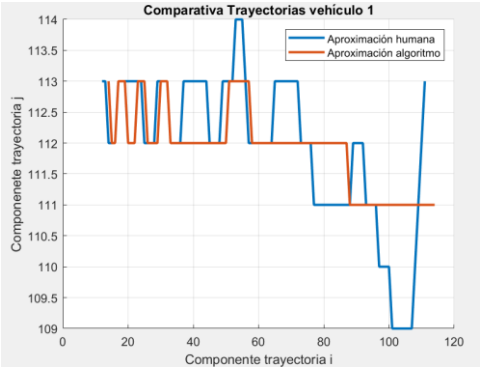

Figura 11-7. Comparativa trayectorias vehículo 1.

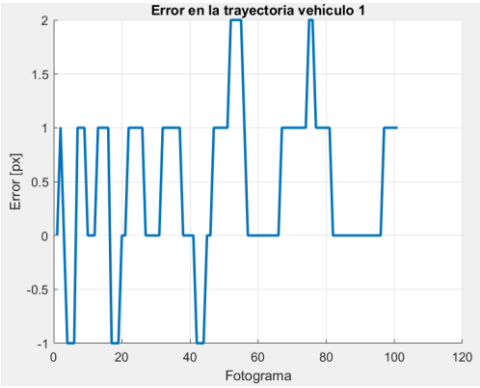

Figura 11-8. Error entre trayectorias vehículo 1.

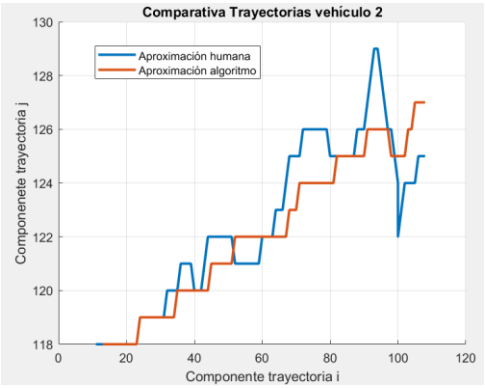

Figura 11-9. Comparativa trayectorias vehículo 2.

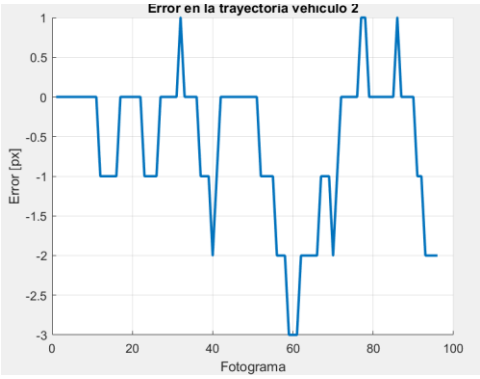

Figura 11-10. Error entre trayectorias vehículo 2.

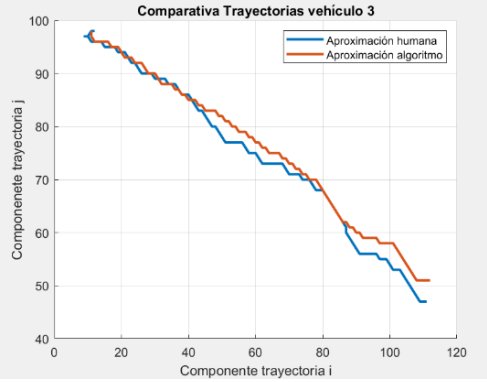

Figura 11-11. Comparativa trayectorias vehículo 3.

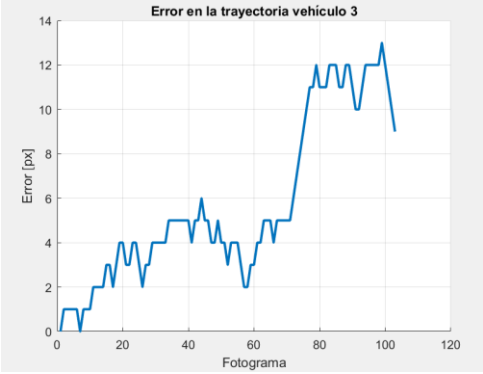

Figura 11-12. Error entre trayectorias vehículo 3.

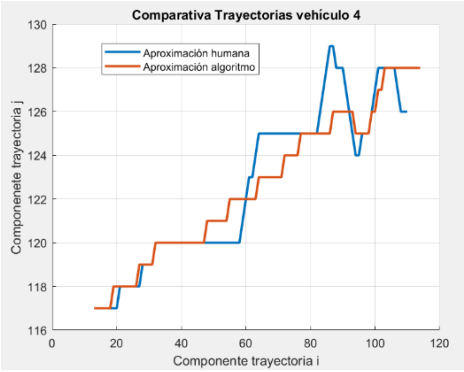

Figura 11-13. Comparativa trayectorias vehículo 4.

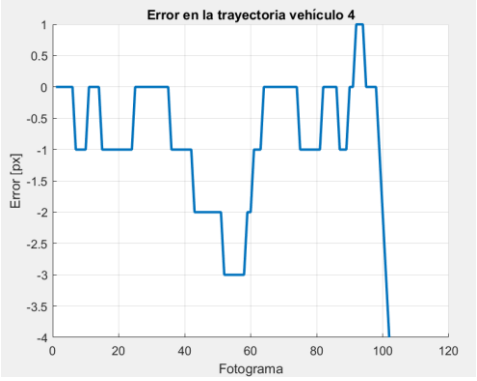

Figura 11-14. Error entre trayectorias vehículo 4.

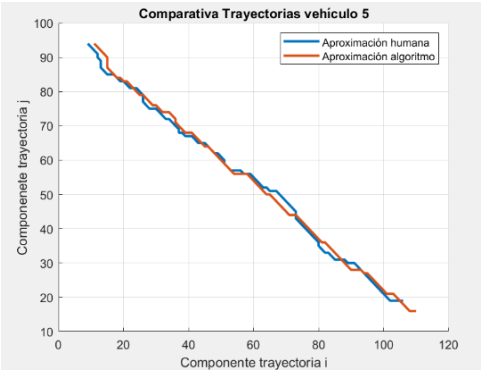

Figura 11-15. Comparativa trayectorias vehículo 5.

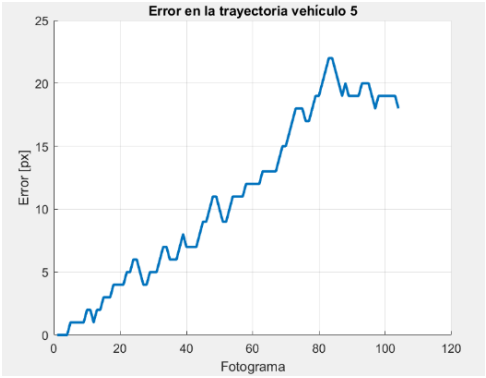

Figura 11-16. Error entre trayectorias vehículo 5.

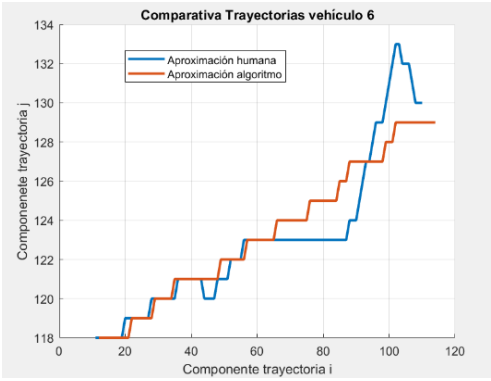

Figura 11-17. Comparativa trayectorias vehículo 6.

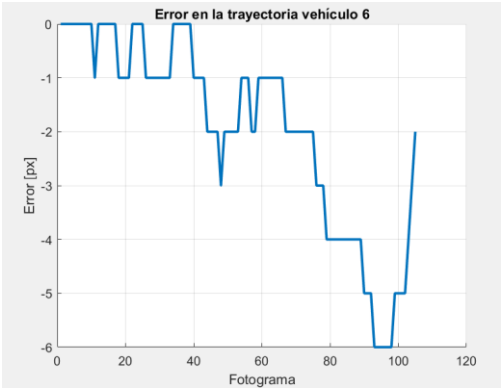

Figura 11-18. Error entre trayectorias vehículo 6.

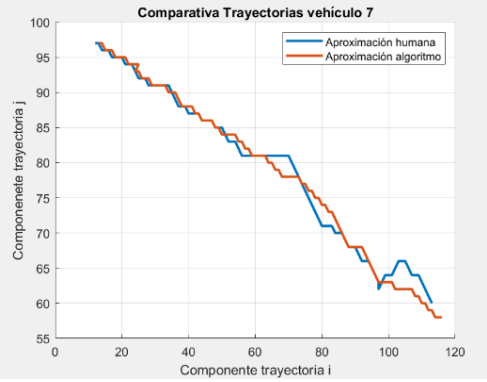

Figura 11-19. Comparativa trayectorias vehículo 7.

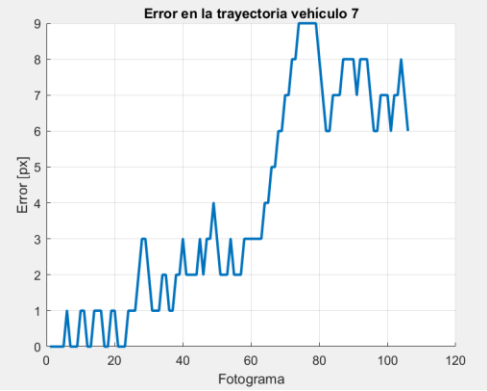

Figura 11-20. Error entre trayectorias vehículo 7.

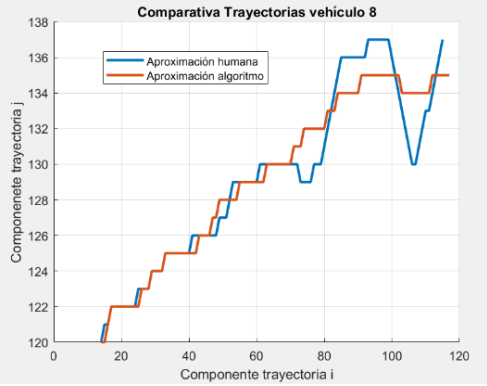

Figura 11-21. Comparativa trayectorias vehículo 8.

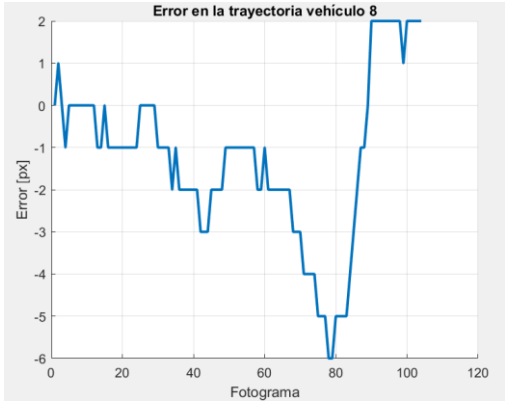

Figura 11-22. Error entre trayectorias vehículo 8.

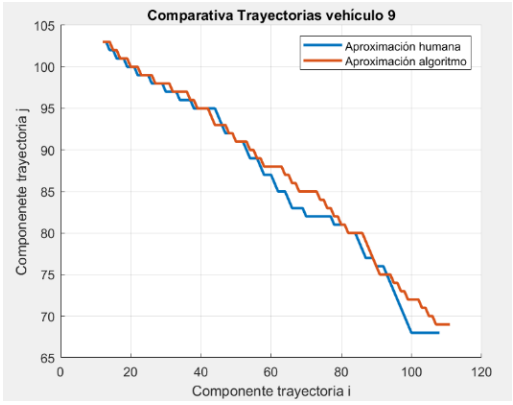

Figura 11-23. Comparativa trayectorias vehículo 9.

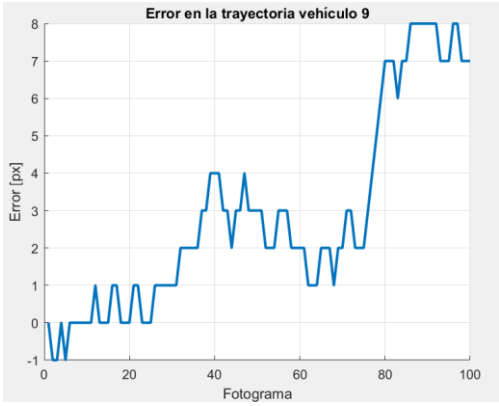

Figura 11-24. Error entre trayectorias vehículo 9.

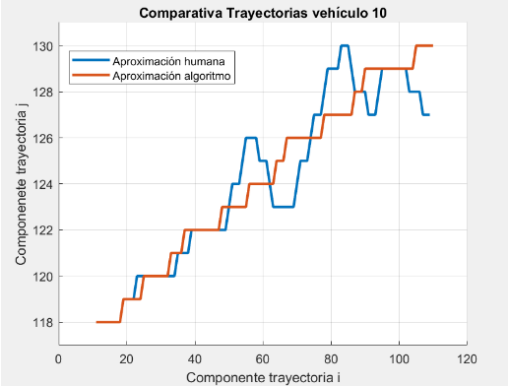

Figura 11-25. Comparativa trayectorias vehículo 10.

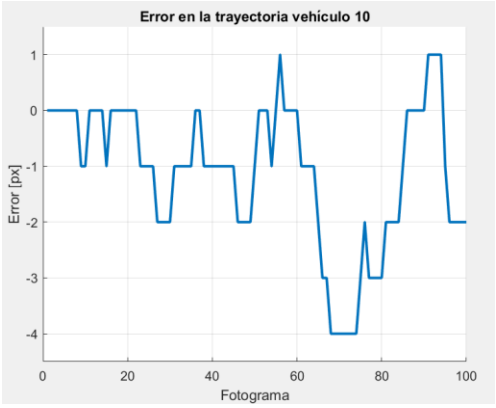

Figura 11-26. Error entre trayectorias vehículo 10.

REFERENCIAS

[1] Nishu singla, Motion Detection Based on Frame Difference Method.

[2] Información Camshift y Meanshift, https://medium.com/@claudio.vindimian/understanding-and-implementing-the-camshift-object-tracking-algorithm-python-81587c24eda8

Temario usado como base para el desarrollo del proyecto:

1. Manuel Ruiz Arahal, Introducción a la estimación del movimiento (temario MIERA).
2. Manuel Ruiz Arahal, Separación de regiones (temario GIERM).

GLOSARIO

SDC: Suma de Diferencias al Cuadrado

Ip: Imagen Procesada

Io: Imagen Original

T: Tensor

i: Fila Imagen

j: Columna Imagen

p: Plantilla

Δi: Delta en filas

Δj: Delta en columnas

Ix: Gradiente en dirección x

Iy: Gradiente en dirección y

λ: Autovalor

c: Vector información punto

for: Bucle matemático

flag: Variable booleana de estado

while: Bucle matemático

double: formato de datos

x: Coordenada longitudinal

y: Coordenada latitudinal