Universidad Zaragoza
1542

# Trabajo Fin de Grado

## Seguimiento de rectas en cámaras omnidireccionales

## Line tracking for fisheye cameras


Autor

Alejandro Castillo Sanjuán

Director

Jesús Bermúdez Cameo

Ponente

José Jesús Guerrero Campo

ESCUELA DE INGENIERÍA Y ARQUITECTURA


2016

# Seguimiento de rectas en cámaras omnidireccionales

## RESUMEN

Debido al gran desarrollo tecnológico de los últimos años, el control y autonomía de vehículos aéreos no tripulados (UAVs) se ha desarrollado enormemente. La localización del vehículo en entornos abiertos se realiza sin problemas usando sistemas de posicionamiento global (GPS). Por contra, en entornos cerrados la señal de GPS no se puede detectar correctamente lo que dificulta la localización y el control autónomo. En estos casos el uso de visión por computador permite utilizar elementos característicos del entorno para localizar el vehículo. La mayoría de estos métodos utilizan características de tipo punto, referenciando una cámara respecto de un mapa que se puede conocer a priori o construir simultáneamente (SLAM). Otro posible tipo de características son las basadas en contornos rectos de la escena. Las características basadas en rectas representan referencias naturales en entornos fabricados por el hombre y a menudo siguen direcciones dominantes, además de ser muy informativas en entornos de interior. Cuando el sistema visual tiene un amplio campo de vista (mediante el uso de cámaras omnidireccionales) los segmentos largos suelen ser completamente visibles, que son especialmente útiles para reducir derivas de localización.

El objeto de este trabajo es la creación de un algoritmo de seguimiento de rectas en cámaras omnidireccionales. Este algoritmo es el primer paso para un método de control autónomo de UAVs en entornos cerrados, como un pasillo.

Para ello se parte de un algoritmo de obtención de líneas en cámaras omnidireccionales con simetría de revolución, y de tres vídeos de un pasillo tomados con una cámara *fisheye*.

Se ha ejecutado dicho algoritmo sobre el primer *frame* de cada vídeo, obteniendo las rectas que se pretenden seguir. El seguimiento de las rectas se realiza mediante el algoritmo de Lucas-Kanade, usado para seguir los puntos que componen dichas rectas. El método propuesto es capaz de gestionar los puntos seguidos de manera errónea, por lo que la recta es recalculada cada cuatro frames para mejorar la estabilidad del seguimiento.

Para la implementación del algoritmo se ha hecho uso del compilador Microsoft Visual Studio en lenguaje C++, y las librerías OpenCV, una biblioteca de uso libre de visión por computador.

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

# DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

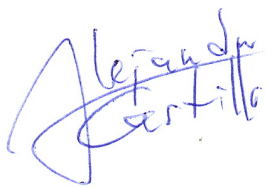
D./Dª. Alejandro Castillo Sanjuán,

con nº de DNI 73158136N en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Seguimiento de rectas en cámaras omnidireccionales,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, Septiembre de 2016



Fdo: Alejandro Castillo Sanjuán

# ÍNDICE

## 1. Introducción

El objeto del presente trabajo es la creación de un algoritmo de seguimiento de rectas en cámaras omnidireccionales, como primer paso a un método de control autónomo de UAVs. Las cámaras omnidireccionales son utilizadas en UAVs [23], como los drones, por su amplio ángulo de visión respecto a las cámaras perspectivas, lo que aporta una mayor información del entorno.

### 1.1 Motivación

Con la gran evolución tecnológica de los últimos años, los vehículos aéreos no tripulados (Unmanned Aerial Vehicle, UAV) han ganado gran fama y autonomía. Esto se puede apreciar en [1], donde el autor recopila el avance que han tenido los vehículos no tripulados desde sus primeras apariciones en 1917 durante la Primera Guerra Mundial, cuando éstos vehículos únicamente se utilizaban para usos militares, hasta la actualidad, donde el número de usos se ha incrementado tanto a nivel militar como civil.

Un ejemplo de estos vehículos son los cuadricópteros, vehículos de pequeño tamaño cuyo uso se ha extendido en áreas tales como la exploración, búsqueda y rescate, topografía, fotografía, agricultura, como juguete de entretenimiento, etc. Éstos aparatos tienen muchas ventajas: debido a su pequeño tamaño poseen una gran portabilidad y maniobrabilidad, además de no arriesgar vidas humanas en tareas como la exploración.

Existen numerosas aplicaciones de los UAVs en las áreas citadas anteriormente. Por ejemplo, en [2] se recoge la utilización de éstos aparatos para la recopilación de imágenes tras desastres naturales tales como el huracán Katrina en 2006 o los terremotos de Haiti en 2010 y Japón en 2011. En [3] se estudia la mejora de las técnicas de búsqueda y rescate con la utilización de UAVs, aprovechándose de la maniobrabilidad de éstos para la localización de las víctimas en zonas difíciles de acceder. Otras aplicaciones se dirigen a la fotografía o grabación, como en [4] donde se presenta la detección y seguimiento de un río mediante un UAV, que puede utilizarse para la grabación de documentales. En el campo de la topografía, en [5] se utilizan para la reconstrucción topográfica de zonas costeras, y en [6] los utiliza para la descripción de zonas dificilmente accesibles, con precisión de centímetros. También son utilizados en la agricultura, donde pueden ser utilizados para el cálculo de la elevación de los campos y la estimación de su biomasa [7] con el objeto de la elaboración de un inventario forestal.

Existen dos tipos de entornos en los que pueden trabajar los cuadricópteros: abiertos y cerrados.

-Entornos abiertos: lugares en espacio abierto, donde no están delimitados por cuatro paredes. En estas situaciones no hay ningún problema en la detección de la posición de los cuadricópteros mediante la señal GPS, así como de su orientación y programación de su recorrido [8].

-Entornos cerrados: lugares que sí están delimitados por cuatro paredes, que se supone que poseen direcciones dominantes, y donde la señal GPS no es detectable, o se detecta con dificultad. Algunos ejemplos de estos escenarios son el pasillo de un edificio, o una calle con edificios altos alrededor sobre los que rebota la señal GPS y dificulta su correcta detección. Estos entornos cerrados son entornos creados por el hombre, en los que se suelen existir muchas líneas paralelas así como unas direcciones principales.

Dadas las dificultades de la detección de la señal GPS en los entornos cerrados, se deberá recurrir a otros medios para que un cuadricóptero recorra autónomamente un entorno de este tipo que resulte desconocido. En estos casos se utiliza la visión artificial para su control. En [9]

hace uso de la visión artificial y flujo óptico para controlar la posición relativa del cuadricóptero sobre la pista de aterrizaje. Otro método es el sistema de localización y modelado simultáneo (SLAM, por las siglas en inglés). El SLAM consiste en un sistema de posicionamiento del vehículo no tripulado en entornos que desconoce, y formando un mapa del exterior a partir de la información que extrae del mismo en tiempo real, actualizando el mapa cada vez que el UAV pasa por una misma zona. En [10] utiliza éste método, pero sólo actualiza el mapa si en él se detectan cambios en una zona que ya ha recorrido previamente, lo que permite unos mejores resultados.

El trabajo [11], combina el SLAM con un algoritmo de reconocimiento de objetos, al que se le proporciona una base de datos con 500 objetos. Al recoger el SLAM información del entorno, cuando localiza uno de los objetos guardados en la base de datos establece en el mapa una marca de referencia en la posición de éste y lo utiliza para guiar al algoritmo de reconocimiento.

### 1.2. Objeto del trabajo

En este trabajo se ha realizado de un algoritmo de seguimiento de rectas en cámaras omnidireccionales. Este algoritmo es el primer paso para un método de control autónomo de UAVs en entornos cerrados basado en la detección de líneas, donde se asume que existen direcciones dominantes características de los entornos creados por el hombre.

El siguiente paso será localizar, entre las rectas detectadas en el entorno en el que el cuadricóptero está situado, la dirección principal en la que interesa que se mueva. Este es un tema que se abordará en un futuro, pero que no entra dentro de los objetivos de este TFG.

La elección de una cámara omnidireccional frente a las cámaras convencionales es que éstas ofrecen un ángulo de visión más amplio: las cámaras ojo de pez más comunes tienen un ángulo de visión de 180° en el eje en el que están enfocadas, frente a los 65° de las cámaras estándar o los 104° de las cámaras de gran angular. Éste ángulo de visión aumenta hasta los 220° en algunas cámaras cuya distancia focal es inferior, formando una imagen circular que incluso se llega a ver la parte de atrás. El tener un campo de visión tan amplio es una ventaja frente a las demás cámaras debido a que captan más información de la imagen que están tomando en cada momento. La desventaja que tienen es que la imagen está distorsionada: las proyecciones de las líneas no son rectas como en las cámaras convencionales, sino que son curvas debido al modelo de proyección no lineal de las cámaras omnidireccionales.

### 1.3. Entorno de trabajo

El algoritmo propuesto en este Trabajo de Fin de Grado en lenguaje C++ utilizando Microsoft Visual Studio 2013 como entorno de desarrollo. Ha sido ejecutado sobre tres vídeos de un mismo pasillo, tomados con una cámara omnidireccional en diferentes posiciones del pasillo.

Las funciones necesarias relativas a la visión por computador se han obtenido mediante las librerías *OpenCV* [12], una biblioteca de uso libre de visión por computador que desde su aparición en el año 2000 se ha convertido en una base para cualquier aplicación de visión por computador.

Se ha contado con la realización de un curso de introducción a la visión por computador [13], donde se han visto algunas de las técnicas utilizadas en este trabajo.

El seguimiento de puntos se ha realizado mediante el algoritmo de Lukas-Kannade, un algoritmo de seguimiento de puntos característicos de la imagen basado en el flujo óptico, que se explicará posteriormente con más detalle.

## 1.4. Estructura del trabajo

Para facilitar la comprensión del lector, se va a explicar la estructura que tiene éste documento.

El trabajo se puede distinguir en tres partes, que siguen el siguiente orden:

1.- Obtención de las rectas que van a ser seguidas a lo largo del vídeo.

2.- Seguimiento de éstas rectas.

3.- Reasignación de puntos que forman las rectas, una vez que se ha perdido un número considerable de éstos debido a fallos en el algoritmo de seguimiento.

Se explicarán en los apartados 2. Extracción de líneas, 3. Detección de movimiento, y 4. Reasignación de los puntos que forman las rectas. Cada una de ellas va acompañada de figuras que facilitan su comprensión.

Posteriormente, en el apartado 5. Algoritmo completo, se explica la combinación de las partes para formar el algoritmo, así como de los resultados obtenidos.

Para finalizar el trabajo se incluye un apartado de conclusiones y trabajo futuro, donde se ofrece la descripción resumida del trabajo realizado, así como de los resultados obtenidos, y el posible trabajo futuro que se puede realizar sobre éste algoritmo.

## 2.- Extracción de las líneas

### 2.1. Cámaras *fisheye*

La cámara utilizada para la extracción de imágenes del entorno es una cámara *fisheye* equiangular. Es una cámara omnidireccional que, como se explica en el apartado 1.2., ofrece un ángulo de visión más amplio, lo que permite extraer más información de la escena al observarse una mayor longitud de las rectas existentes en ella.

La causa de que con éstas cámaras se pueda obtener una mayor información es que la imagen obtenida está deformada. En las figuras siguientes se puede ver, de forma simplificada y en dos dimensiones, la toma de un punto de la imagen por una cámara perspectiva y por otra omnidireccional, siendo:

-X, el punto del entorno que toma la cámara.

-x, el punto proyectado en la imagen.

-O, centro óptico de la cámara.

-f, distancia focal.

-Z y R, profundidad y radio del punto X en coordenadas cilíndricas..

En la primera de las imágenes se muestra la proyección de un punto por una cámara perspectiva. Un punto tridimensional de la escena X se proyecta a través del centro óptico O en el punto de la imagen x.

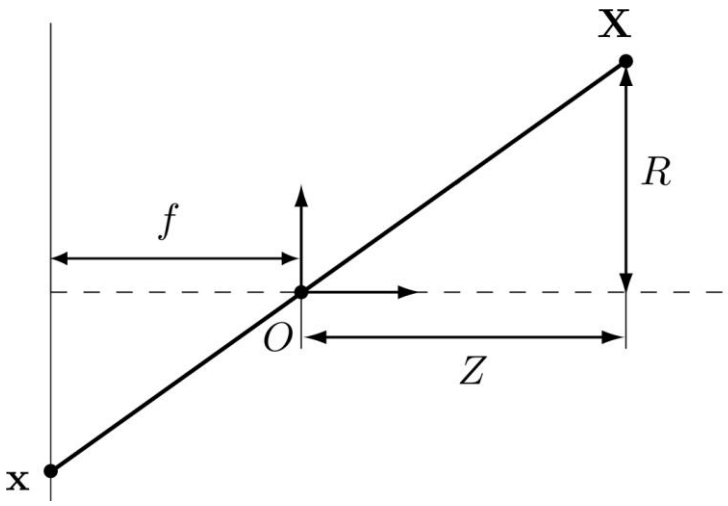



**Figura 1.** Proyección de un punto en cámara perspectiva

En la segunda se muestra cómo una cámara omnidireccional proyecta el mismo punto. La distorsión provocada por la lente "fisheye" equiangular es equivalente a proyectar el punto X sobre imagen que envuelve a una semiesfera.

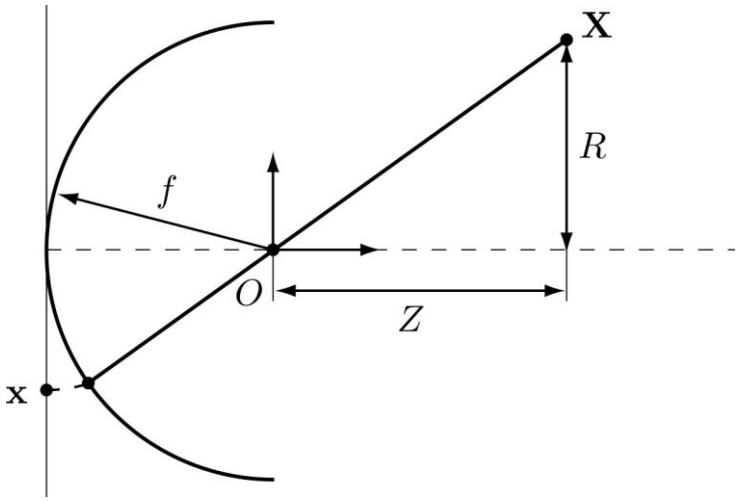



**Figura 2.** Proyección de un punto en una cámara omnidireccional.

La diferencia entre las distintas proyecciones de un mismo punto también se pueden observar en las ecuaciones que definen la posición del punto en la imagen que toma la cámara, mostradas en la figura 3. Ésta imagen es de dos dimensiones, por lo que la posición del punto está formada por dos coordenadas x e y, siendo el origen de coordenadas el centro de la imagen. Su posición en el entorno, al ser un espacio de tres dimensiones, se determina por tres coordenadas X, Y Z, siendo el origen de coordenadas la posición de la cámara. El parámetro R es la distancia euclídea del punto X al eje de simetría de revolución de la cámara, y de valor $R = \sqrt{X^2 + Y^2}$ .

Existen varios tipos de cámaras omnidireccionales, que distorsionan la imagen en mayor o menor medida. En este caso se muestran las ecuaciones correspondientes a una cámara omnidireccional equiangular, que es el tipo que se ha utilizado para este trabajo.

| | **Coordenada X** | **Coordenada Y** |
|---|---|---|
| **Proyección de un punto en cámara perspectiva** | $x = f\,\frac{X}{Z}$ | $y = f\,\frac{Y}{Z}$ |
| **Proyección de un punto en cámara *fisheye* equiangular** | $x = f\,\arctan\left(\frac{R}{Z}\right)\frac{X}{R}$ | $y = f\,\arctan\left(\frac{R}{Z}\right)\frac{Y}{R}$ |

**Figura 3.** Modelo de proyección de un punto en cámaras perspectiva y omnidireccional

Una vez vista la diferencia de la proyección de los puntos en ambas cámaras, pasamos a estudiar la proyección de líneas. En cámaras omnidireccionales, la proyección 2D de las líneas 3D pertenecientes al entorno no se ajusta a una recta, como ocurre en las cámaras convencionales y en las cuáles la cuáles la obtención de su proyección es relativamente sencilla, sino que es una curva y por tanto su modelo matemático no es lineal. En la figura siguiente se muestran las ecuaciones que definen una recta, en un modelo perspectivo y en otro omnidireccional equiangular, siendo $r = \sqrt{x^2 + y^2}$ , y $n_x$ , $n_y$ y $n_z$ las componentes del

vector normal del plano del entorno que contiene a la recta y al punto en el que se sitúa la cámara (ver figura 12).

| | |
|---|---|
| **Recta en cámara perspectiva** | $n_x x + n_y y - n_z f = 0$ |
| **Recta en fisheye equiangular** | $n_x x + n_y y + n_z r \cot\left(\frac{r}{f}\right) = 0$ |

**Figura 4.** Modelo de proyección de una recta en cámaras perspectiva y omnidireccional

En la siguientes figuras se puede observar cómo el modelo omnidireccional deforma la imagen que es tomada por la cámara.

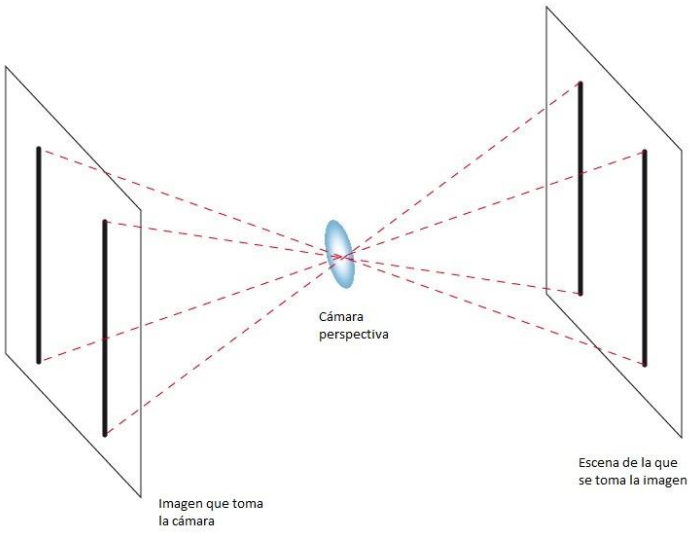



**Figura 5.** Proyección de imagen con una cámara perspectiva

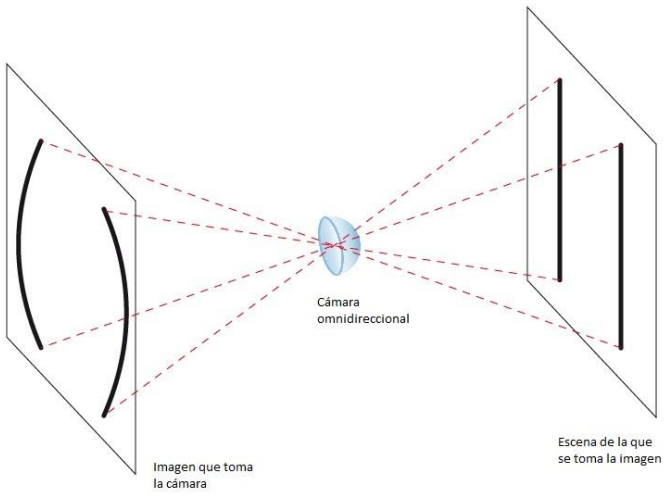



**Figura 6.** Proyección de imagen con una cámara omnidireccional

## 2.2. Algoritmo de extracción de líneas en cámaras omnidireccionales

El algoritmo utilizado para la extracción de las proyecciones de las rectas que se van a seguir es el descrito en [14], donde extrae los planos de proyección de las rectas de la imagen omnidireccional, quedando definidas las rectas por la normal del plano que las contiene.

El funcionamiento es el siguiente: una vez obtenida la imagen por medio de una cámara omnidireccional, se pasa a blanco y negro y se le aplica el algoritmo de Canny [15].Este es un algoritmo para hallar los contornos existentes en la imagen, que pese a tener un coste computacional más alto que otros métodos de detección de bordes, asegura que sólo tienen un píxel de ancho, lo que evita fallos de detección. Los puntos detectados como bordes se separan en contornos, y se almacena cada contorno por separado. Sólo se almacenan los contornos que contienen un número de puntos superior a un umbral (en este caso el algoritmo trabaja con un umbral de 200 puntos). Una vez obtenidos estos puntos, se le aplica un algoritmo RANSAC, un algoritmo iterativo para la obtención de modelos matemáticos y que se explica su funcionamiento en el apartado 2.3, a los puntos de cada contorno. Al aplicarse a contornos con un número de puntos superior al umbral, se asegura que el número de datos es suficiente para un correcto funcionamiento de éste algoritmo. De esta forma se obtiene la normal del plano que contiene la proyección de la recta que forma cada contorno y al punto en el que la cámara está situada siendo ésta normal del plano lo que define cada recta. Los puntos que se ajustan a los planos de proyección hallados se guardan en vectores, conteniendo cada vector los puntos correspondientes a un plano de proyección, para su posterior seguimiento.


**Figura 7.** Diagrama de flujo del algoritmo de extracción de líneas

Las siguientes figuras muestran la aplicación del algoritmo de extracción de rectas a un *frame* de uno de los vídeos en los que se ha ejecutado el algoritmo objeto de este trabajo. Se puede observar la no linealidad de las proyecciones, así como que éstas van hacia tres puntos de fuga, correspondientes a las tres direcciones principales existentes en los entornos construidos por el hombre.

**Figura 8.** *Frame* omnidireccional.

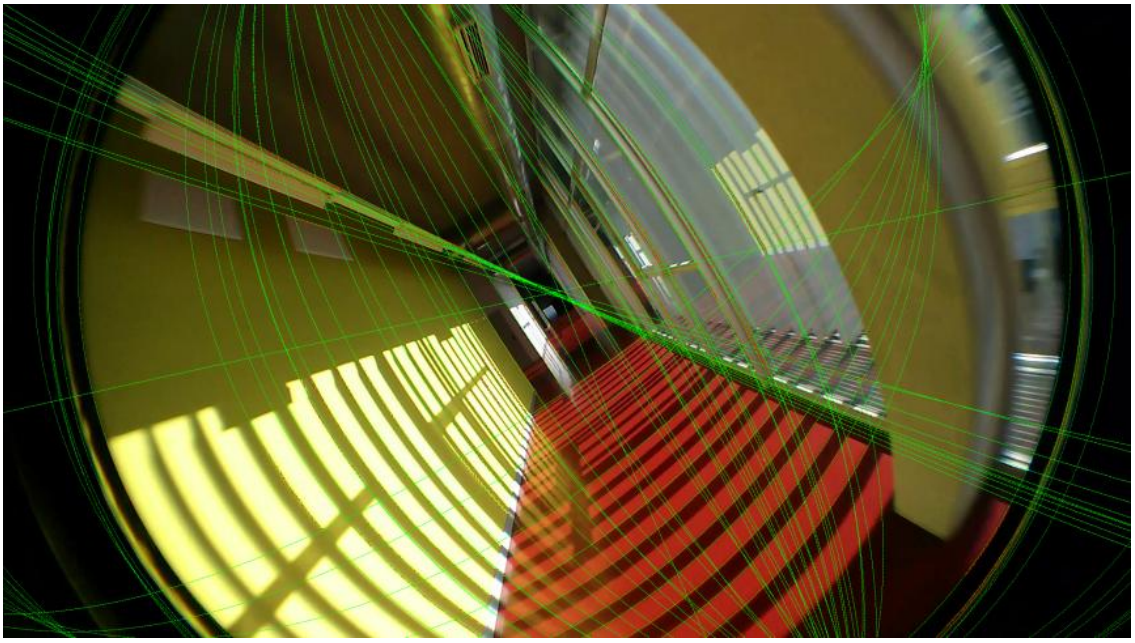


**Figura 9.** Aplicación del algoritmo de extracción de líneas.

## 2.3. RANSAC

Este algoritmo está basado en el método RANSAC (abreviación de RANdom SAmple Consensus) para la obtención de las rectas. Es un algoritmo iterativo de estimación que permite hallar un modelo matemático a partir de una muestra de datos que contiene una cantidad considerable de datos atípicos. Fue publicado por primera vez en 1981 por Fischler y Bolles [16], y desde entonces se ha utilizado en numerosas aplicaciones de análisis de imágenes.

Para simplificar su explicación se pondrá como ejemplo el ajuste de una recta a partir de un conjunto de puntos, del que se quiere obtener la recta que mejor se ajusta a este conjunto. El método tiene los siguientes pasos:

1.- Se eligen dos puntos al azar, y se traza la recta que éstos puntos forman.

2.- Se observa cuántos de los puntos se ajustan al modelo (recta).

3.- Se eligen otros dos puntos al azar, y se traza la recta que forman.

4.- Se vuelve a observar los puntos que se ajustan a esta nueva recta.

5.- Se repiten los pasos 1 al 4, y se escoge como solución la recta a la que más puntos se han ajustado.

Los puntos que se ajustan al modelo matemático elegido como solución son llamados *inliers*, mientras que los que no se ajustan al modelo son *outliers*.

Es necesario realizar un número mínimo de iteraciones, k, que viene definido por la fórmula:

$$k = \frac{\log(1-p)}{\log(1-w^s)}$$

Siendo:

-s es el número mínimo de puntos a partir del cual el modelo matemático buscado puede ser estimado (s=2, en el caso de buscar una recta).

-p es la probabilidad de que al elegir al azar un subconjunto de puntos s, todos sean *inlier*.

-w es la probabilidad de que un punto del conjunto sea un *outlier*.

Este método de ajuste de rectas tiene como ventaja que no se tienen en cuenta los datos atípicos presentes en el conjunto de puntos a muestrear, realizándose un ajuste más preciso de la recta que si se realizara por mínimos cuadrados. En las siguientes figuras se puede observar el ajuste de una línea a partir de un conjunto de datos, en el que existen datos atípicos, mediante el método de mínimos cuadrados y el método RANSAC. En éste segundo método se observa cómo la línea se ajusta perfectamente a un número de puntos (*inliers*), mientras que no tiene en cuenta los valores atípicos (*outliers*).

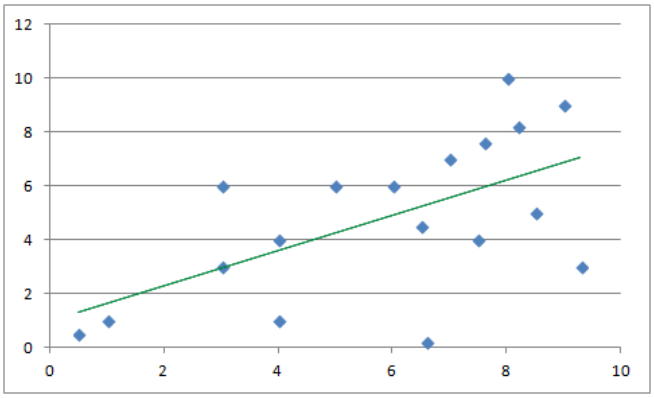



**Figura 10.** Ajuste de la línea por mínimos cuadrados

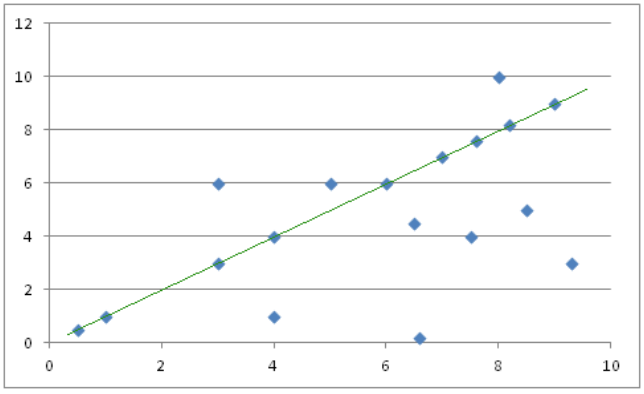



**Figura 11.** Ajuste de la línea por RANSAC

El método RANSAC aplicado en este algoritmo sigue el mismo funcionamiento que el ejemplo simplificado explicado anteriormente, salvo que en este caso el modelo matemático buscado es un plano. Para definir un plano son necesarios tres puntos: uno de ellos será el punto en el que se sitúa la cámara, y los otros dos serán pertenecientes a uno de los contornos de la imagen.

La figura 12 muestra, de forma simplificada en dos dimensiones, cómo se forma éste plano con los tres puntos. El punto O es en el que se sitúa la cámara, mientras que los puntos A y B están contenidos en un contorno que forma la línea L. El plano que contiene a todos ellos queda definido por la normal n.

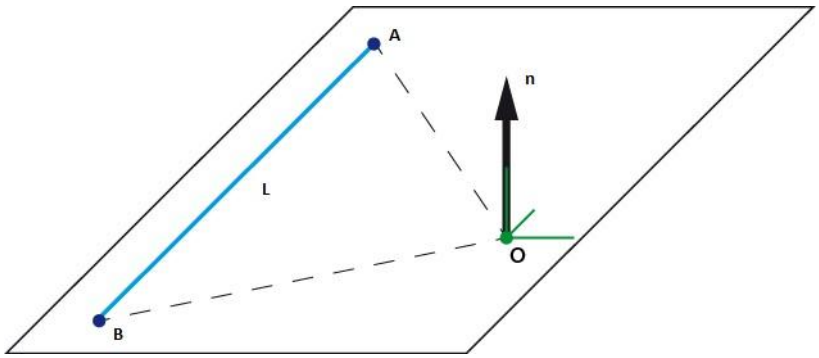



**Figura 12.** Plano formado por el punto de la cámara y dos del contorno.

Los pasos a seguir son similares a los ya explicados, salvo por algunos cambios debidos a que el modelo matemático que se va a hallar es un plano y no una recta:

1.- Se eligen dos puntos al azar, y con el punto origen, se halla el plano que forman. Éste es el nuevo modelo matemático que se quiere hallar: en este caso los dos puntos que se eligen aleatoriamente forman planos, no rectas, junto con el punto origen de planos.

2.- Se aplica el método RANSAC a cada contorno por separado: cada contorno genera una proyección de una recta.

3.- Los puntos que se ajustan a cada plano elegido como solución se almacenan, para posteriormente realizar un seguimiento de estos puntos, y por tanto de las proyecciones de las rectas que forman, a lo largo de un video.

En la figura 13, obtenida de [14], se observa una imagen omnidireccional en la que un plano ha sido definido por dos puntos de un contorno, pintados en verde. El punto azul, al ajustarse al plano, es un *inlier*, mientras que el rojo es un *outlier* al no pertenecer a éste modelo matemático.

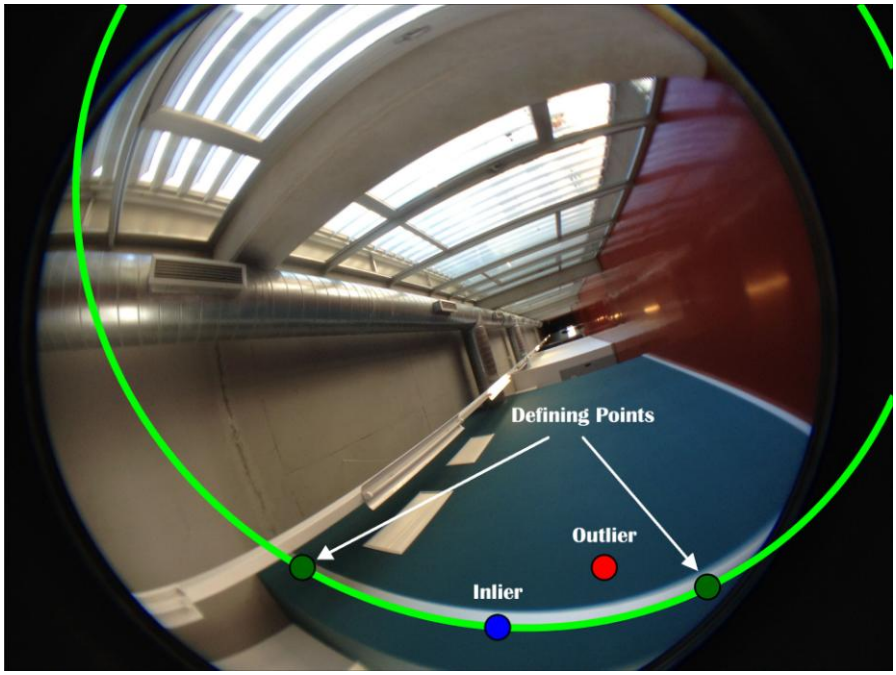



**Figura 13.** Aplicación del RANSAC a la obtención de planos.

## 2.4. Aplicación del algoritmo al trabajo

En este trabajo, se ejecuta el algoritmo de extracción de rectas al primer *frame* del vídeo, para obtener las líneas que van a ser seguidas. Los puntos de la imagen que las forman se guardan en vectores de puntos, que serán seguidos mediante un algoritmo de seguimiento de puntos que se explica posteriormente en el apartado 3.2.

## 3.- Detección de movimiento

### 3.1. Introducción a la detección de movimiento

El análisis de la detección de movimiento está relacionado, entre otros, con aplicaciones en tiempo real de detección y seguimiento de objetos, con la obtención de información de los objetos estáticos y en movimiento de una escena de trabajo, para poder tomar una serie de decisiones.

Se aplican las siguientes restricciones:

- Cambios de luz en la escena pueden asignar movimiento a objetos estáticos. Por tanto, se asume que no existen cambios de iluminación entre una imagen y otra.

- Se asume que los objetos son rígidos, y no están sometidos a deformaciones durante el movimiento.

Existen dos distintos algoritmos de detección de movimiento: cámara estática y flujo óptico. Éste último es el que se utiliza en este trabajo, y por tanto es el que más se va a explicar su funcionamiento.

En el algoritmo de cámara estática, como su propio nombre indica, la cámara que toma el vídeo a evaluar tiene una posición fija. Ésta es una restricción adicional a las ya comentadas anteriormente: la cámara permanece inmóvil en todo momento, y son los objetos los que están en movimiento. Esto produce que el escenario en el que transcurre el video a evaluar no varíe en toda su duración, detectando únicamente el movimiento de los objetos presentes en la escena.

Un método muy utilizado es el de diferencia de imágenes. La aplicación de este método consiste en la resta entre una imagen patrón, previamente almacenada, y las distintas imágenes de un vídeo. De esta forma se puede detectar el movimiento fácilmente en determinadas regiones de interés de la imagen. Tiene la desventaja de ser un método simple y poco preciso, pero es rápido y tiene un gran rango de aplicaciones.

En la siguiente figura, obtenida de [13], podemos ver un ejemplo de la aplicación de este método a una secuencia de imágenes, tomadas con una cámara estática sobre el mismo fondo. Sobre este fondo aparecen personas, que estarán en movimiento. Además también se detectara la variación de posición de las banderas ocasionada por el viento que las mueve. En la tercera imagen se puede observar la aplicación del método de la diferencia de imágenes, una vez segmentada. Los píxeles en blanco se corresponden a los objetos en movimiento, en nuestro caso tanto las personas como las banderas, mientras que los píxeles negros corresponden al fondo que se mantiene estático. Podemos observar pequeños puntos blancos que no se corresponden con ningún objeto en movimiento, y que son ocasionados por fallos en el algoritmo de detección de movimiento.

**Figura 14.** Detección de movimiento con cámara estática.

Como hemos dicho, este es un método rápido y con un gran rango de aplicaciones, pero tiene algunas limitaciones:

-La cámara debe estar estática.

-Indica en qué zonas existe movimiento, pero no da ninguna información acerca de la magnitud o dirección de la velocidad de movimiento.

En el flujo óptico, u optical flow en lengua anglosajona, no requiere de la estaticidad de la cámara, sino que identifica regiones de la imágenes que están en movimiento. Se basa en el cambio de intensidad de la imagen en una secuencia de imágenes. Con el flujo óptico es posible obtener lo que se llama un mapa de velocidades, un conjunto de vectores que nos dice la dirección y la magnitud del movimiento. Este conjunto de vectores nos muestra el movimiento relativo de los píxeles de una imagen. De esta forma, evitamos depender de una imagen estática, pues podemos identificar en el mapa de movimiento aquellos que son producidos por el movimiento de la cámara, y de esta forma poder descartarlos de la detección de movimiento.

Existen dos tipos de flujo óptico: flujo óptico denso, y flujo óptico esparcido. El primero se realiza sobre toda la imagen, lo que le da la ventaja de que no requiere de unas características específicas en la imagen para seguir el movimiento. Un ejemplo es el algoritmo de Horn-Schunk [17], donde asume que el campo de velocidades varía suavemente en toda la imagen. En cambio, en el segundo tipo lo que se hace es buscar en dos imágenes consecutivas cuál es la zona más parecida de la segunda imagen al entorno de un punto seleccionado de la primera, lo que hace que se pueda trabajar más rápido. El método más utilizado en el flujo óptico esparcido es el de Lucas-Kanade[18], que se explicará con más detalle más adelante.

En la siguiente imagen, obtenida de [19], se puede observar el flujo óptico que genera una esfera que gira en sentido horario.

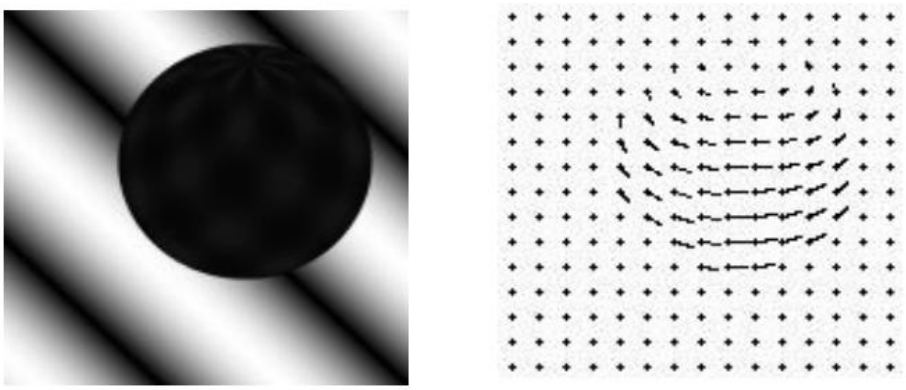


**Figura 15.** Flujo óptico de una esfera al girar

Un problema a tener en cuenta en este método es el llamado problema de la apertura. Es causado por la información limitada que tenemos cuando un objeto no es observado en su totalidad. Se va a explicar este problema con un ejemplo:

En la figura 16 se puede ver la imagen que toma la cámara en un momento dado (imagen izquierda), siendo el círculo la imagen que obtiene la cámara, y la parte negra un objeto que localiza. Se puede observar que la cámara sólo detecta una parte del objeto, proporcionándonos solo una parte de la información de éste.

En la misma figura, en la imagen derecha se observa la imagen que toma la cámara de éste objeto unos instantes después. Se puede apreciar que el objeto se ha movido, pero al no poder apreciar el objeto en su totalidad no se puede afirmar si el objeto se ha desplazado hacia abajo, o se ha acercado hacia la cámara.

**Figura 16.** Problema de la apertura.

Por tanto, en la práctica será necesario imponer restricciones adicionales para la detección del movimiento de objetos.

### 3.2. Lucas-Kanade

Como se ha dicho anteriormente, uno de los algoritmos más utilizados en seguimiento de objetos es el llamado método de Lucas-Kanade. Este algoritmo es un ejemplo de flujo óptico esparcido: no calcula todos los vectores de velocidad para cada píxel de la imagen, sino que únicamente lo hace en regiones de la imagen que son más sencillas de detectar.

Estas regiones suelen ser puntos característicos de los objetos a seguir, como bordes o esquinas, que se denominan puntos de interés. Son obtenidos aplicándole a la imagen un algoritmo de detección de detección de éstos puntos. Unos de los más utilizados son el detector de Harris [20], que detecta esquinas y bordes, y el algoritmo de detección de esquinas de Shi-Tomasi [21], que se centra más en la detección de esquinas de la imagen.

Una vez obtenidas las regiones a seguir, se realiza el flujo óptico asumiendo tres condiciones:

-Constancia en el brillo: Un píxel perteneciente a un objeto no cambiará de apariencia entre los imágenes consecutivas.

-Persistencia temporal: los movimientos de los objetos cambian lentamente con el tiempo.

-Coherencia espacial: los puntos de un objeto que están próximos tienen el mismo movimiento. Ésta es la principal característica de este algoritmo: asume que el desplazamiento de dos puntos cercanos entre dos imágenes consecutivas es pequeño, lo que se conoce como el criterio de vecindad. Permite asumir que el flujo óptico en una región pequeña es constante.

Esta última condición limita el algoritmo a aplicaciones donde los puntos a seguir estén próximos entre sí en dos imágenes consecutivas, lo que implica que la velocidad de los objetos debe ser lo suficientemente lenta como para que la detección se realice correctamente. En otras palabras, la velocidad de la cámara en movimiento es limitada.

Este algoritmo siempre se encuentra en forma de pirámide de la imagen con varios niveles: En el primer nivel se aplica a la imagen sin aplicarle ningún tipo de filtro, es decir, la imagen evaluada es la diferencia entre la imagen del video y la siguiente. En el segundo se le aplica previamente a la imagen un filtro gaussiano para reducir su resolución antes de procesar la imagen con el algoritmo de Lucas-Kanade. En el tercero se le vuelve a aplicar el filtro gaussiano a la imagen del segundo nivel, y así sucesivamente. La imagen resultado es la suma de los niveles calculados. En la figura 16 muestra un esquema del funcionamiento del algoritmo, con una pirámide de cuatro niveles. Se puede observar su similitud a una pirámide, de ahí que la implementación del algoritmo se le conozca como Lucas-Kanade piramidal [22].

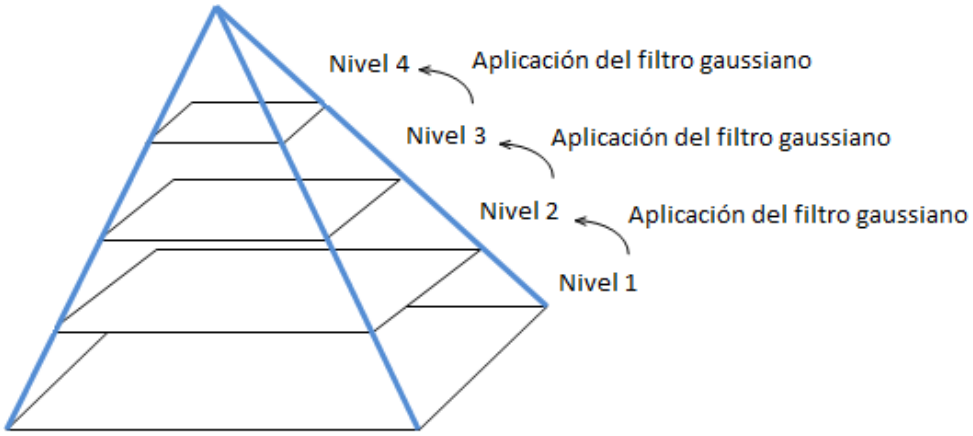



**Figura 17.** Lucas-Kanade piramidal

Con este método se consiguen detectar los más mínimos movimientos. Esto también es un problema, pues se pueden detectar pequeñas variaciones de los píxeles que son debidas a una mala captación de la imagen. También pueden dar errores cuando el movimiento de la cámara es grande, o cuando el brillo no es constante.

### 3.3. Aplicación de la detección de movimiento al trabajo

Como se ha comentado en la explicación de la detección de movimiento, el método utilizado para este trabajo es el de Lucas-Kanade, debido a que la cámara utilizada para grabar el vídeo sobre el que se va a trabajar está en movimiento. Este algoritmo es conocido y muy fiable para el seguimiento de puntos característicos, como esquinas o manchas, pero en este caso se va a utilizar para seguir los puntos de los contornos que forman las rectas cuyo seguimiento es el objeto de este trabajo. Esto puede ocasionar que el seguimiento de los puntos no sea totalmente correcto, ya que las regiones próximas de los puntos del contorno pueden ser muy parecidas, o incluso prácticamente iguales. En nuestro caso esto no es un problema, pues el objeto del trabajo no es el de seguir los puntos , sino las proyecciones de las rectas que éstos puntos forman. Por tanto, no importa el que en su seguimiento varíe ligeramente la posición de los puntos como consecuencia de un fallo en su seguimiento, pues si esta variación no es muy grande el conjunto de puntos seguidos seguirá creando la proyección de la recta sin ningún problema.

Si la posición de un punto en dos *frames* consecutivos varía una longitud mayor a un umbral, éste punto se elimina del conjunto de puntos que van a crear la proyección de la recta. La selección de este umbral debe hacerse con cuidado, pues si es demasiado pequeño, intentando eliminar al máximo los fallos cometidos, ningún punto del contorno tendrá su posición en el siguiente *frame* a una distancia inferior al umbral, y por tanto ninguno de ellos será tomado en cuenta en el siguiente *frame*.

Esto se va a mostrar con dos ejemplos de dos situaciones, una vez aplicadas el algoritmo de detección de bordes de Canny, que es cómo se realiza en este trabajo, por ello están en blanco y negro:

-En el primero de ellos se ve el borde de un objeto. Aplicando un algoritmo de búsqueda de puntos característicos, previo al seguimiento de éstos puntos, el punto a seguir sería la esquina de este objeto. En el *frame* siguiente se ve cómo se ha desplazado, y la trayectoria que ha seguido éste punto.

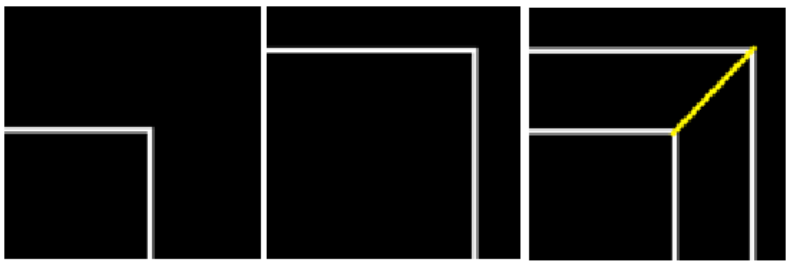


**Figura 18.** Seguimiento de esquinas con Lucas-Kanade

-En este segundo ejemplo, se ve el borde de una pared, tomado con una cámara omnidireccional, en dos *frames* consecutivos. Al ejecutar sobre el primer *frame* el algoritmo de extracción de las proyecciones de las líneas en cámaras omnidireccionales, se obtendrían los

puntos obtenidos por este algoritmo a lo largo del contorno, y su posición en el segundo *frame* tras el seguimiento realizado con el algoritmo de Lucas-Kanade. Se puede ver como algunos de los puntos no están en la posición que deberían estar, pero siguen estando sobre el contorno, por tanto el conjunto de puntos seguirá creando la proyección de la línea a seguir. Se observa también que uno de los puntos no sigue sobre el contorno, debido a un fallo de seguimiento. Este punto se eliminará del conjunto de puntos que forman la línea, y no será tenido en cuenta en el siguiente frame al variar su posición más del límite especificado.

**Figura 19.** Seguimiento de los puntos de la recta con Lucas-Kanade.

Como puede observarse, en cada *frame* se pierden algunos puntos de cada contorno debido a un fallo en su seguimiento. Para asegurar un correcto seguimiento de las rectas, debe realizarse una reasignación de los puntos contenidos en los contornos cuando se pierda un número suficiente de éstos. Lo ideal sería que ésta reasignación se produjera a cada *frame*, para que la pérdida de puntos fuese lo mínima posible, pero el coste computacional sería muy elevado. Por tanto, se reasignan nuevos puntos a los contornos cada cuatro *frames*. El procedimiento realizado para llegar a esta conclusión se explica en el anexo 1.

**4. Reasignación de los puntos que forman las rectas**

Tras lo explicado en el apartado anterior, se observa que a cada *frame* del vídeo se pierden algunos de los puntos que forman las líneas. El número de puntos perdidos no es el mismo en todos los *frames*, sino que depende de los fallos que tenga el algoritmo de seguimiento cada vez que se realiza.

Al perderse puntos prácticamente en la totalidad de los *frames*, la formación de las líneas se verá afectada negativamente. Como se ha explicado en el apartado 2.3, el algoritmo RANSAC funciona mejor cuantos más puntos hay en la muestra.

Para asegurar en todo momento la correcta formación de las proyecciones a seguir, es necesario introducir más puntos a cada contorno que se ajusten a las líneas. Estos nuevos puntos serán aquellos que sean detectados como bordes y se ajusten a la línea, pero que no están en el vector de puntos que se ajusta a cada contorno.

Para ello, se procede de la siguiente forma:

1.- Se pasa un algoritmo de Canny para hallar los bordes existentes en el *frame*. De esta forma obtenemos una imagen binaria: los puntos que pertenecen a un borde son blancos, mientras que los que no pertenecen son negros.

2.- Todos los puntos de tipo "borde", son almacenados en un vector.

3.- Se calcula la distancia de todos estos puntos a las líneas que están siendo seguidas. Si la distancia es inferior a un valor umbral, todos los puntos que cumplan esta condición serán los nuevos puntos que formarán el vector de puntos que se ajusta a esa línea.

4.- Los nuevos vectores de puntos son los seguidos en los siguientes *frames*.

Esto se puede ver en las siguientes figuras:

En la primera se ha representado la imagen del *frame* tras haberle ejecutado el algoritmo de Canny, donde se puede distinguir los puntos que son bordes pintados en blanco sobre un fondo negro, y sobre la que se han pintado las líneas que se están siguiendo en rojo.

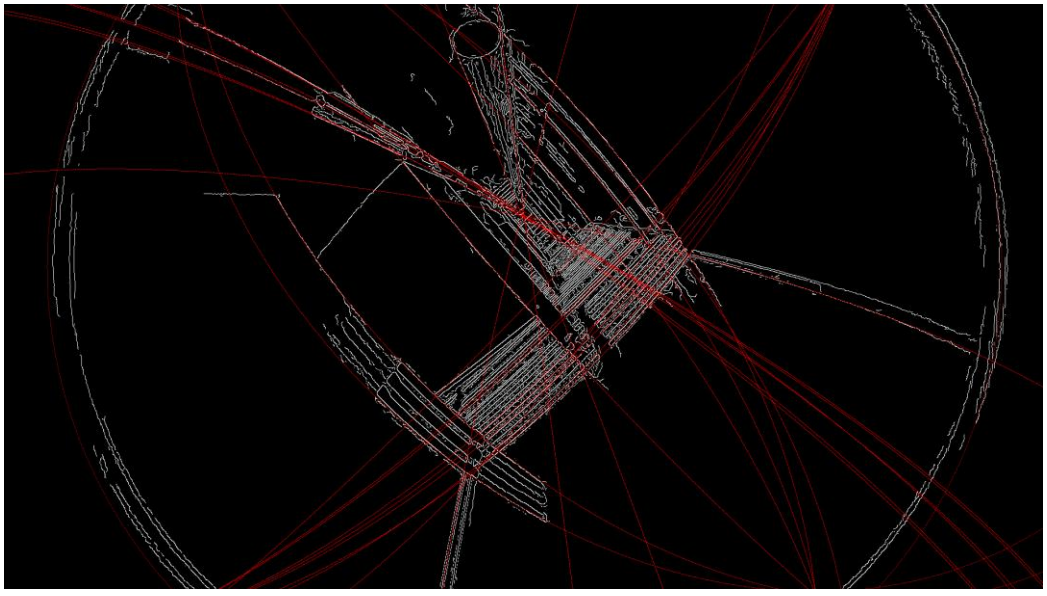


**Figura 20.** Aplicación del algoritmo de Canny sobre el *frame*.

En la segunda se muestran sólo los puntos que son asignados a los vectores de los contornos para su posterior seguimiento. Se puede observar que, además de los puntos pertenecientes al contorno, se han asignado otros puntos que se ajustan al plano que contiene la proyección de la línea, pero que no están en ese contorno. Es por esto que, tras la reasignación, el número de puntos de cada contorno es notablemente superior al que había en el *frame* anterior, lo que permite continuar siguiendo la línea sin problemas. En el anexo 2 se muestra el número de puntos de los contornos antes y después de la reasignación.

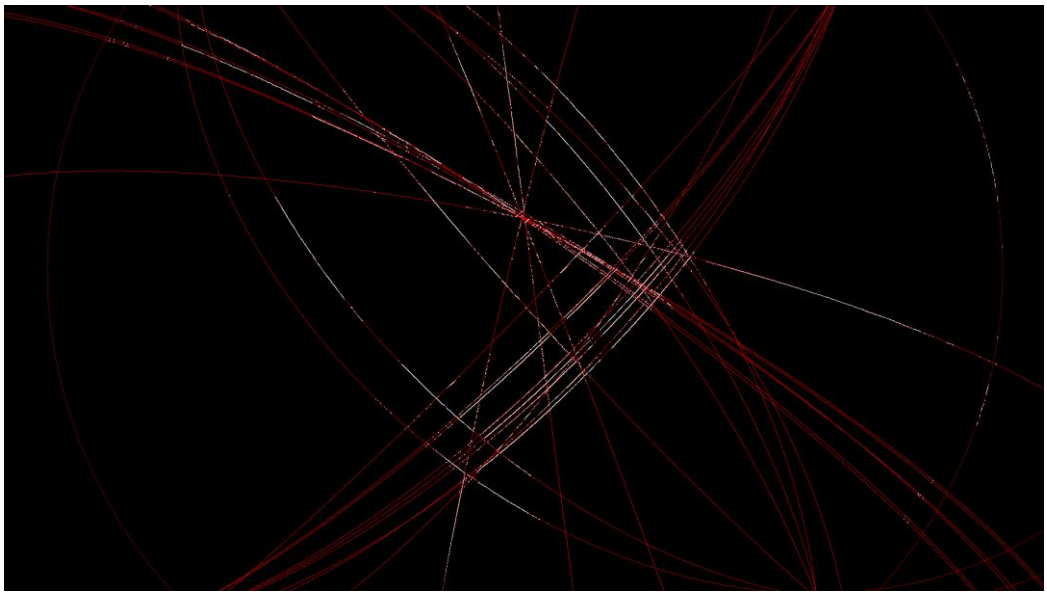


**Figura 21.** Reasignación de puntos a los contornos.

## 5.- Algoritmo completo

Una vez explicadas las partes más importantes de éste algoritmo, se va a explicar la combinación realizada de estos dos métodos para la ejecución del programa, así como la solución encontrada para la reasignación de los puntos.

El programa se puede explicar siguiendo los siguientes pasos, ilustrando cada uno de ellos con una imagen. Cada uno de estos pasos corresponde a una de las partes en las que se divide la estructura del algoritmo.

### 5.1.Extracción de las rectas

En primer lugar, se lleva a cabo la ejecución del algoritmo de extracción de las proyecciones de las rectas al primer *frame* del vídeo. Con esto, se hallan los puntos que forman las proyecciones, y son los que se van a seguir para llevar a cabo el seguimiento de las rectas. Se almacenan en un vector de vectores de puntos, conteniendo cada vector de puntos los correspondientes a la proyección de una línea (es decir, a un contorno).

En la figura 22 se puede observar la aplicación del algoritmo al primer *frame*, donde están pintados en verde los puntos de los contornos que forman las rectas, y en rojo las rectas que se van a seguir.

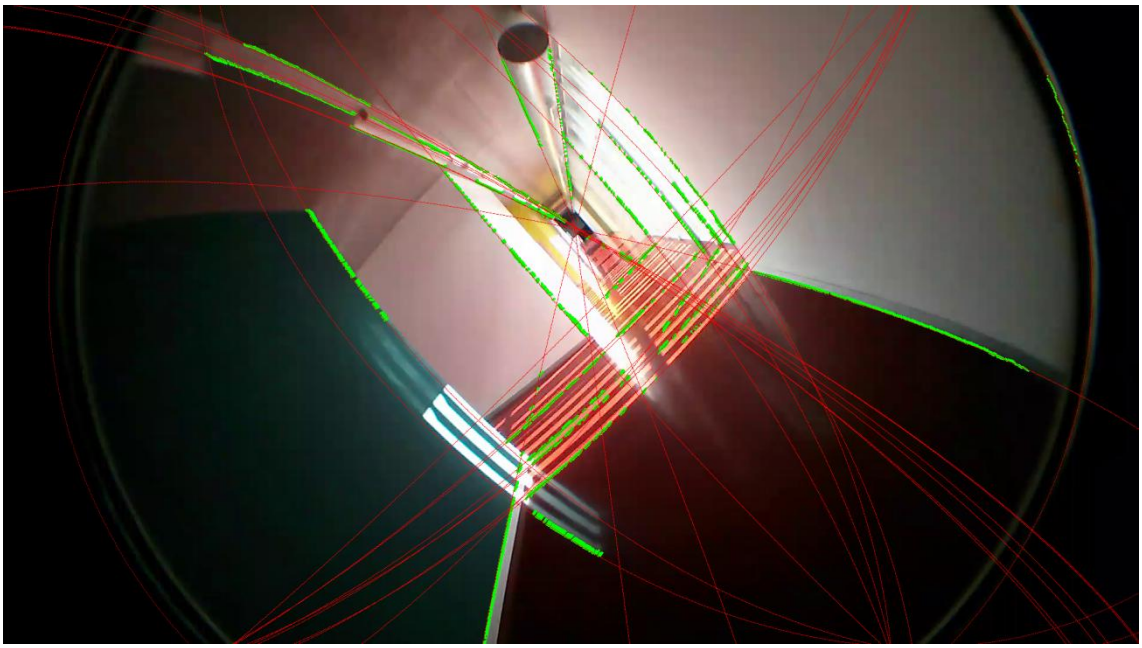


**Figura 22.** Obtención de las líneas de la imagen, y de los puntos que las forman

### 5.2. Detección de movimiento

Se realiza el seguimiento de cada vector de puntos de cada contorno por separado, mediante el algoritmo de Lucas-Kanade. Como se ha explicado en el apartado 3.2, este algoritmo de seguimiento puede ocasionar fallos en el seguimiento de puntos. Es por esto que se establece un umbral, una distancia máxima entre las posiciones de un mismo punto en dos imágenes consecutivas. Si la distancia es menor al umbral, se considera que el seguimiento del punto se

ha realizado correctamente, mientras que si es superior el seguimiento del punto no es válido, y se elimina del vector de puntos que lo contenía.

Este umbral se ha establecido siguiendo el siguiente criterio:

-El movimiento de la cámara es limitado, por tanto la posición de un mismo punto en dos *frames* consecutivos varía mínimamente.

-Como bien se explica en el apartado citado anteriormente, puede existir un fallo en el seguimiento de puntos del contorno, pero que éste no influya en la formación de la proyección de la recta. Se considerará que esto ocurre si la "desviación" del punto es inferior al umbral.

En la imagen siguiente se observa un *frame* del vídeo (por simplicidad no se han pintado las rectas a seguir), en el que se ha unido mediante una línea verde la posición de cada punto en el *frame*, con su posición calculada en el *frame* siguiente.

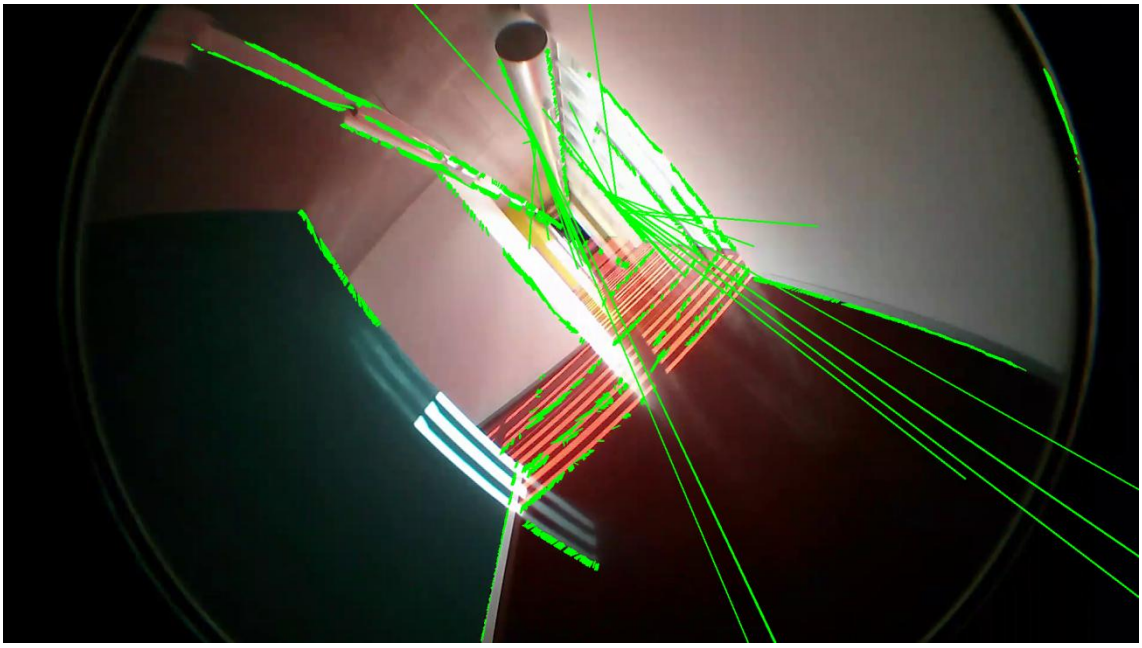


**Figura 23.** Seguimiento de los puntos

Como puede verse, las líneas que unen algunos de los puntos son muy largas, incluso se salen de la imagen. Esto es ocasionado por los fallos de seguimiento comentados anteriormente, y los puntos en los que ocurre esto se eliminarán del vector de puntos del contorno en el que estén.

### 5.3. Reasignación de puntos

Los fallos en el seguimiento de puntos ocasionan la pérdida de éstos, por lo que se realiza su reasignación cada cuatro *frames*. Como se explica en el anexo 2, tras la reasignación, en cada contorno se recogen los puntos que son bordes y que se ajustan a la proyección de la recta seguida. Esto se puede ver en las siguientes imágenes, en las que muestra la misma región de la imagen en el tercer y cuarto *frame* del primer vídeo. Para una mejor visualización, no se han pintado las líneas que forman los contornos, tan sólo los puntos.

**Figura 24.** Diferencia de puntos entre el tercer *frame* y el cuarto.

Se puede ver como el número de puntos en la región de la derecha es muy superior. Aparecen nuevos puntos en bordes que en la imagen de la izquierda no se consideran en el vector de contornos. Éstos son los nuevos puntos que se han reasignado a los contornos, que pese a no pertenecer al contorno que forma la proyección de la línea al que están siendo asignados, se ajustan a ésta proyección.

Como ya se ha dicho en el apartado 2.3, el que existan más puntos en la muestra sobre la que se va a ejecutar un algoritmo RANSAC es beneficioso para su ejecución, pues el modelo matemático buscado estará más ajustado.

La reasignación de puntos se realiza cada cuatro *frames*. En ese momento, los contornos recuperan puntos para poder seguir creando la proyección de las rectas. Pero es posible que un contorno pierda todos sus puntos por un mal seguimiento en los *frames* entre dos reasignaciones, o que contenga un número de puntos por debajo de un umbral establecido como el número mínimo de puntos para considerar que ese contorno puede seguir formando una línea de forma fiable. Este valor umbral se ha establecido en 20 puntos. En ese caso, se considera que se ha perdido la línea que forma ese contorno, y no se seguirá en los siguientes *frames* del vídeo. Para los contornos que contienen un número de puntos por debajo del umbral para formar la línea, se le asigna un vector de ceros para mostrar que esa línea se ha dejado de seguir.

La pérdida de puntos ocasiona que algunas de las rectas no sean seguidas correctamente. Esto es debido a que los puntos originales que forman el contorno son perdidos por un fallo de seguimiento, pero la línea se sigue formando porque pasa por otros bordes, de los que extrae puntos que guarda en su vector de puntos, y éstos son los que la forman.

Esto se puede ver en la figura 25. Es una imagen tomada en un *frame* del vídeo 1, en el que se puede observar el mal seguimiento de las líneas, pintadas en rojo. Algunas de ellas, como las formadas por las ventanas en la parte derecha de la imagen, no se ajustan a los contornos que las han formado inicialmente.

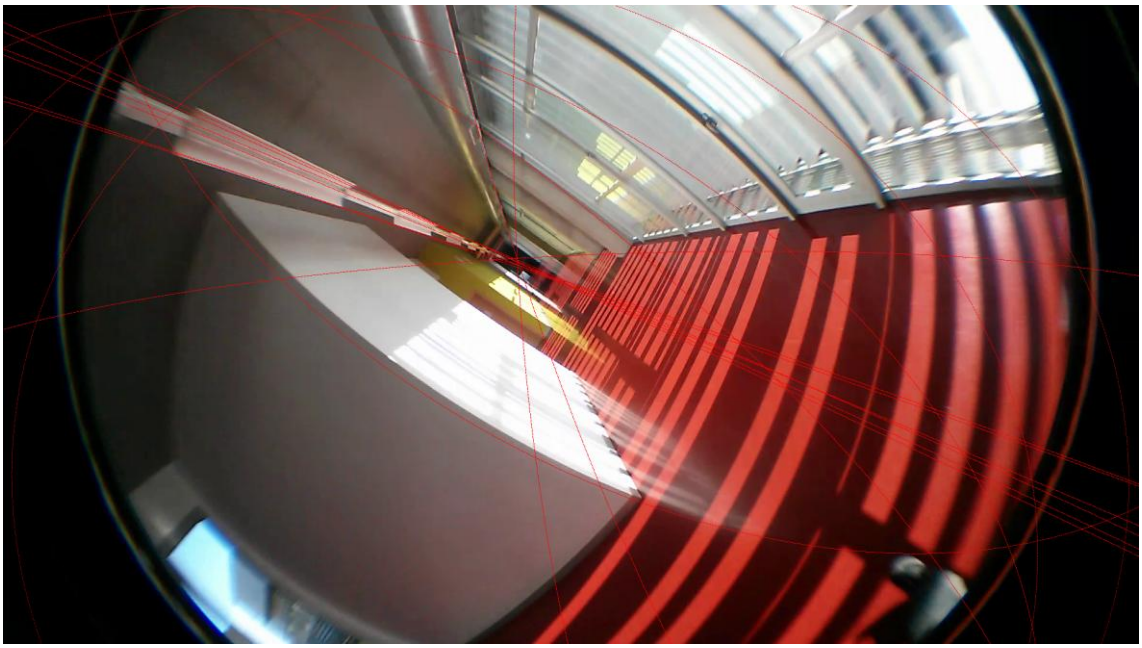


**Figura 25.** Seguimiento de líneas con umbral 20 puntos.

En el caso de subir el umbral del mínimo de puntos, se pierde un mayor número de líneas, pero se asegura un mejor seguimiento. En la figura 26 podemos ver el mismo *frame* del vídeo anterior, habiendo subido el umbral a 60 puntos. Se muestra como se han eliminado las líneas que se seguían incorrectamente, además de alguna que no se ajustaba correctamente al contorno, como la esquina de la pared blanca de la izquierda.

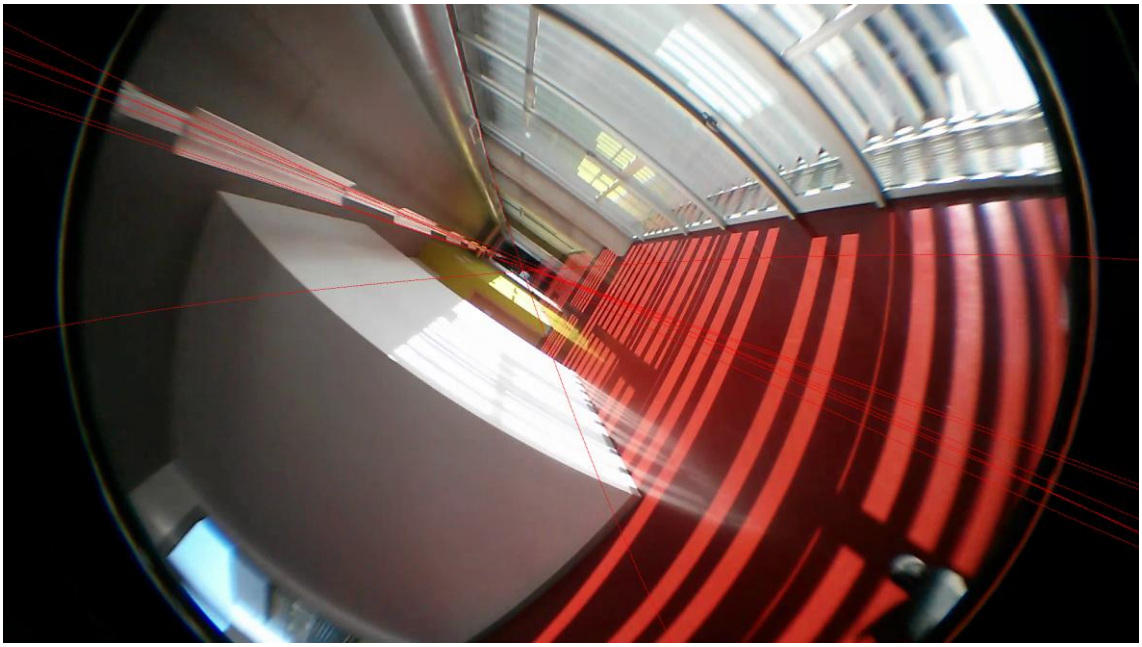


**Figura 26.** Seguimiento de líneas con umbral 60 puntos.

5.4. Representación gráfica del algoritmo

La representación gráfica del algoritmo completo se muestra en el diagrama de flujo de la figura 26. De esta manera se puede observar el funcionamiento del programa de una forma más visual, lo que puede ayudar a su comprensión. Con la intención de simplificar su realización el diagrama se ha realizado unificando el algoritmo de extracción de líneas en un sólo bloque, dado que no es el objeto de este trabajo.

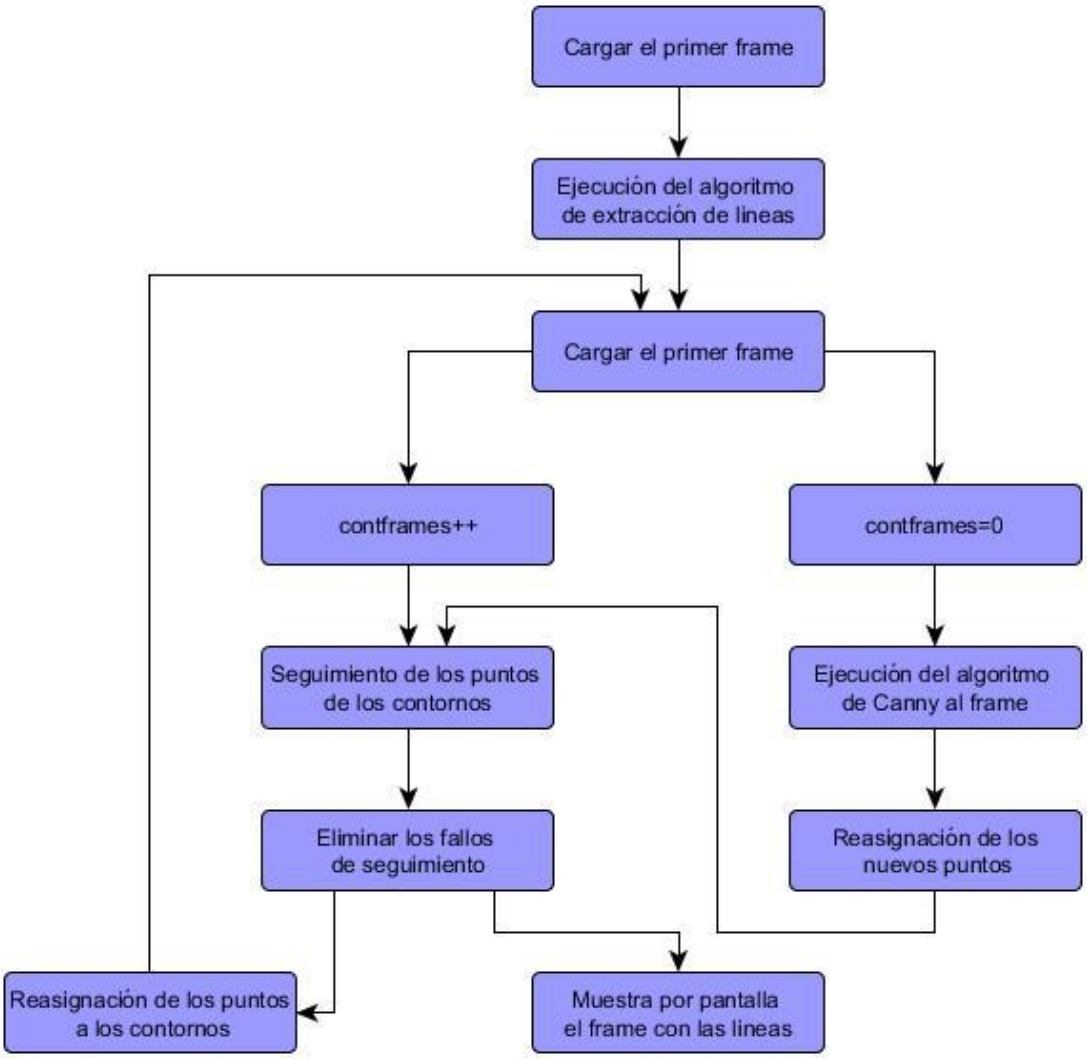



**Figura 27.** Diagrama de flujo del programa.

## 6. Conclusiones y trabajo futuro

En este trabajo, se ha realizado un algoritmo de seguimiento de rectas en cámaras omnidireccionales. Dicho algoritmo se ha realizado en lenguaje C++, y se ha utilizado el compilador Microsoft Visual Studio para su desarrollo.

El seguimiento de las rectas se ha realizado siguiendo los puntos que las forman mediante el algoritmo de Lucas-Kanade piramidal, un método muy utilizado en el seguimiento de puntos basado en el flujo óptico. En ocasiones, éste método ofrece fallos en el seguimiento de los puntos, por lo que se han eliminado los puntos que se considera que su seguimiento ha resultado fallido. Al eliminar estos fallos, se producen muchas pérdidas de puntos, que se convierten en pérdidas de rectas si éstas llegan a tener un número de puntos menor al valor umbral establecido. Cuanto más alto es ese valor, más precisa es la recta que forman, pero más rectas se pierden por contener puntos por debajo de ese umbral.

Se llega a un conflicto: al bajar el umbral de puntos mínimos, se produce una menor pérdida de rectas, pero se compromete la calidad del seguimiento de éstas. Por otra parte, si se sube este umbral, el seguimiento de rectas es más fiable, pero el número de pérdidas de rectas aumenta.

El algoritmo se ha evaluado únicamente en tres vídeos tomados en el mismo pasillo, el cual está bien iluminado, y la velocidad a la que se mueve la cámara es lenta y constante. Para mejorar el programa, se podría ejecutar en vídeos en los cuales la iluminación no sea la adecuada, o a distinta velocidad, para observar el comportamiento del algoritmo en estos entornos.

Otro posible aspecto a mejorar del algoritmo es el método tras el que se obtiene que la reasignación de puntos debe ser cada cuatro *frames*. Se podría realizar una asignación dinámica, cuando la diferencia del número de puntos en los contornos en un momento dado respecto a los que había inicialmente fuese superior a un umbral.

Este algoritmo sigue las rectas obtenidas tras la ejecución del algoritmo de extracción de rectas. Aplicando este algoritmo a un pasillo, las rectas obtenidas siguen las direcciones principales de éste. Por tanto, se podría seguir desarrollando para conseguir implementar un sistema de navegación autónomo para UAVs en entornos cerrados, en el que se detecte las líneas seguidas que se ajustan a la dirección principal que se quiere seguir.

## ANEXO 1. Frame en el que se produce la reasignación

Para llegar la conclusión de que la reasignación de puntos debe realizarse cada cuatro *frames*, se ha observado el número de puntos que tiene cada contorno en cada uno de los *frames*, en los tres vídeos en los que se ha evaluado el algoritmo. En las siguientes tablas, se muestra el número de puntos que contienen los contornos tras el seguimiento de sus puntos en cada *frame*, en cada uno de los vídeos evaluados, así como la diferencia en tanto por ciento de los puntos que contiene un contorno respecto del *frame* inicial.

**Tabla 1.** Número de puntos en los contornos en el vídeo 1.

| contorno | frame 0 | frame 1 | diferencia | frame 2 | diferencia | frame 3 | diferencia | frame 4 | diferencia | frame 5 | diferencia | frame 6 | diferencia | frame 7 | diferencia |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 142 | 142 | 0,00 | 142 | 0,00 | 142 | 0,00 | 142 | 0,00 | 142 | 0,00 | 141 | 0,70 | 141 | 0,70 |
| 1 | 165 | 165 | 0,00 | 165 | 0,00 | 144 | 12,73 | 141 | 14,55 | 136 | 17,58 | 136 | 17,58 | 136 | 17,58 |
| 2 | 145 | 145 | 0,00 | 145 | 0,00 | 115 | 20,69 | 106 | 26,90 | 103 | 28,97 | 103 | 28,97 | 103 | 28,97 |
| 3 | 140 | 140 | 0,00 | 140 | 0,00 | 117 | 16,43 | 100 | 28,57 | 100 | 28,57 | 96 | 31,43 | 96 | 31,43 |
| 4 | 144 | 144 | 0,00 | 144 | 0,00 | 140 | 2,78 | 105 | 27,08 | 103 | 28,47 | 103 | 28,47 | 103 | 28,47 |
| 5 | 248 | 246 | 0,81 | 246 | 0,81 | 245 | 1,21 | 242 | 2,42 | 242 | 2,42 | 242 | 2,42 | 242 | 2,42 |
| 6 | 579 | 579 | 0,00 | 579 | 0,00 | 579 | 0,00 | 577 | 0,35 | 573 | 1,04 | 565 | 2,42 | 565 | 2,42 |
| 7 | 108 | 108 | 0,00 | 108 | 0,00 | 108 | 0,00 | 108 | 0,00 | 108 | 0,00 | 108 | 0,00 | 108 | 0,00 |
| 8 | 104 | 104 | 0,00 | 104 | 0,00 | 92 | 11,54 | 89 | 14,42 | 87 | 16,35 | 87 | 16,35 | 87 | 16,35 |
| 9 | 104 | 104 | 0,00 | 104 | 0,00 | 104 | 0,00 | 101 | 2,88 | 101 | 2,88 | 101 | 2,88 | 101 | 2,88 |
| 10 | 109 | 109 | 0,00 | 109 | 0,00 | 109 | 0,00 | 106 | 2,75 | 106 | 2,75 | 106 | 2,75 | 106 | 2,75 |
| 11 | 201 | 201 | 0,00 | 201 | 0,00 | 180 | 10,45 | 145 | 27,86 | 106 | 47,26 | 101 | 49,75 | 101 | 49,75 |
| 12 | 206 | 206 | 0,00 | 206 | 0,00 | 205 | 0,49 | 206 | 0,00 | 206 | 0,00 | 206 | 0,00 | 206 | 0,00 |
| 13 | 59 | 59 | 0,00 | 59 | 0,00 | 59 | 0,00 | 59 | 0,00 | 57 | 3,39 | 57 | 3,39 | 57 | 3,39 |
| 14 | 137 | 137 | 0,00 | 137 | 0,00 | 136 | 0,73 | 136 | 0,73 | 128 | 6,57 | 100 | 27,01 | 100 | 27,01 |
| 15 | 237 | 235 | 0,84 | 235 | 0,84 | 163 | 31,22 | 123 | 48,10 | 98 | 58,65 | 97 | 59,07 | 97 | 59,07 |
| 16 | 191 | 191 | 0,00 | 191 | 0,00 | 191 | 0,00 | 191 | 0,00 | 191 | 0,00 | 191 | 0,00 | 191 | 0,00 |
| 17 | 143 | 143 | 0,00 | 143 | 0,00 | 139 | 2,80 | 127 | 11,19 | 125 | 12,59 | 107 | 25,17 | 107 | 25,17 |
| 18 | 252 | 252 | 0,00 | 252 | 0,00 | 252 | 0,00 | 233 | 7,54 | 212 | 15,87 | 195 | 22,62 | 195 | 22,62 |
| 19 | 196 | 196 | 0,00 | 196 | 0,00 | 196 | 0,00 | 186 | 5,10 | 182 | 7,14 | 182 | 7,14 | 182 | 7,14 |
| 20 | 131 | 131 | 0,00 | 131 | 0,00 | 129 | 1,53 | 75 | 42,75 | 46 | 64,89 | 35 | 73,28 | 35 | 73,28 |
| 21 | 245 | 243 | 0,82 | 243 | 0,82 | 169 | 31,02 | 152 | 37,96 | 126 | 48,57 | 114 | 53,47 | 114 | 53,47 |
| 22 | 220 | 220 | 0,00 | 220 | 0,00 | 220 | 0,00 | 176 | 20,00 | 158 | 28,18 | 117 | 46,82 | 117 | 46,82 |
| 23 | 354 | 354 | 0,00 | 349 | 1,41 | 339 | 4,24 | 209 | 40,96 | 179 | 49,44 | 121 | 65,82 | 121 | 65,82 |
| 24 | 319 | 317 | 0,63 | 316 | 0,94 | 316 | 0,94 | 198 | 37,93 | 171 | 46,39 | 120 | 62,38 | 120 | 62,38 |
| 25 | 174 | 174 | 0,00 | 174 | 0,00 | 174 | 0,00 | 174 | 0,00 | 172 | 1,15 | 172 | 1,15 | 172 | 1,15 |

Con los datos de la tabla, se puede observar que los fallos en los dos primeros *frames* no son significativos. En el tercero ya algunos contornos sufren fallos de consideración: los contornos 15 y 21 han perdido el 30% de sus puntos. Pero es en el cuarto en el que se producen fallos importantes: prácticamente la totalidad de los contornos han perdido puntos, algunos de ellos

casi la mitad de los que inicialmente tenían. Por ello, sería en este *frame* en el que se debería realizar la reasignación.

**Tabla 2.** Número de puntos en los contornos en el vídeo 2.

| contorno | frame 0 | frame 1 | diferencia | frame 2 | diferencia | frame 3 | diferencia | frame 4 | diferencia | frame 5 | diferencia | frame 6 | diferencia | frame 7 | diferencia |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 215 | 215 | 0,00 | 215 | 0,00 | 215 | 0,00 | 215 | 0,00 | 215 | 0,00 | 4 | 98,14 | 0 | 100,00 |
| 1 | 200 | 200 | 0,00 | 200 | 0,00 | 198 | 1,00 | 190 | 5,00 | 176 | 12,00 | 8 | 96,00 | 0 | 100,00 |
| 2 | 170 | 170 | 0,00 | 170 | 0,00 | 168 | 1,18 | 167 | 1,76 | 167 | 1,76 | 167 | 1,76 | 167 | 1,76 |
| 3 | 161 | 161 | 0,00 | 161 | 0,00 | 161 | 0,00 | 161 | 0,00 | 161 | 0,00 | 158 | 1,86 | 158 | 1,86 |
| 4 | 121 | 121 | 0,00 | 106 | 12,40 | 106 | 12,40 | 106 | 12,40 | 104 | 14,05 | 34 | 71,90 | 29 | 76,03 |
| 5 | 105 | 104 | 0,95 | 90 | 14,29 | 90 | 14,29 | 90 | 14,29 | 90 | 14,29 | 30 | 71,43 | 30 | 71,43 |
| 6 | 115 | 115 | 0,00 | 115 | 0,00 | 115 | 0,00 | 115 | 0,00 | 115 | 0,00 | 44 | 61,74 | 44 | 61,74 |
| 7 | 241 | 241 | 0,00 | 240 | 0,41 | 225 | 6,64 | 225 | 6,64 | 223 | 7,47 | 48 | 80,08 | 48 | 80,08 |
| 8 | 151 | 141 | 6,62 | 141 | 6,62 | 138 | 8,61 | 138 | 8,61 | 138 | 8,61 | 52 | 65,56 | 52 | 65,56 |
| 9 | 112 | 112 | 0,00 | 112 | 0,00 | 112 | 0,00 | 112 | 0,00 | 112 | 0,00 | 67 | 40,18 | 67 | 40,18 |
| 10 | 134 | 134 | 0,00 | 134 | 0,00 | 134 | 0,00 | 134 | 0,00 | 134 | 0,00 | 110 | 17,91 | 110 | 17,91 |
| 11 | 107 | 106 | 0,93 | 106 | 0,93 | 106 | 0,93 | 104 | 2,80 | 99 | 7,48 | 21 | 80,37 | 21 | 80,37 |
| 12 | 218 | 218 | 0,00 | 205 | 5,96 | 205 | 5,96 | 205 | 5,96 | 205 | 5,96 | 0 | 100,00 | 0 | 100,00 |
| 13 | 214 | 214 | 0,00 | 214 | 0,00 | 214 | 0,00 | 214 | 0,00 | 214 | 0,00 | 63 | 70,56 | 63 | 70,56 |
| 14 | 163 | 163 | 0,00 | 163 | 0,00 | 163 | 0,00 | 163 | 0,00 | 163 | 0,00 | 117 | 28,22 | 117 | 28,22 |
| 15 | 245 | 243 | 0,82 | 199 | 18,78 | 199 | 18,78 | 199 | 18,78 | 199 | 18,78 | 11 | 95,51 | 0 | 100,00 |
| 16 | 175 | 175 | 0,00 | 175 | 0,00 | 175 | 0,00 | 175 | 0,00 | 175 | 0,00 | 30 | 82,86 | 30 | 82,86 |
| 17 | 142 | 142 | 0,00 | 142 | 0,00 | 142 | 0,00 | 142 | 0,00 | 142 | 0,00 | 123 | 13,38 | 123 | 13,38 |
| 18 | 154 | 154 | 0,00 | 154 | 0,00 | 154 | 0,00 | 154 | 0,00 | 154 | 0,00 | 154 | 0,00 | 154 | 0,00 |
| 19 | 95 | 95 | 0,00 | 95 | 0,00 | 95 | 0,00 | 95 | 0,00 | 95 | 0,00 | 95 | 0,00 | 95 | 0,00 |
| 20 | 77 | 76 | 1,30 | 76 | 1,30 | 76 | 1,30 | 76 | 1,30 | 76 | 1,30 | 47 | 38,96 | 47 | 38,96 |
| 21 | 306 | 306 | 0,00 | 302 | 1,31 | 300 | 1,96 | 300 | 1,96 | 300 | 1,96 | 52 | 83,01 | 50 | 83,66 |
| 22 | 164 | 163 | 0,61 | 159 | 3,05 | 157 | 4,27 | 156 | 4,88 | 147 | 10,37 | 144 | 12,20 | 140 | 14,63 |
| 23 | 62 | 62 | 0,00 | 62 | 0,00 | 62 | 0,00 | 62 | 0,00 | 62 | 0,00 | 61 | 1,61 | 61 | 1,61 |
| 24 | 105 | 105 | 0,00 | 105 | 0,00 | 105 | 0,00 | 105 | 0,00 | 105 | 0,00 | 58 | 44,76 | 58 | 44,76 |
| 25 | 175 | 175 | 0,00 | 175 | 0,00 | 175 | 0,00 | 175 | 0,00 | 175 | 0,00 | 145 | 17,14 | 145 | 17,14 |
| 26 | 129 | 129 | 0,00 | 129 | 0,00 | 129 | 0,00 | 129 | 0,00 | 129 | 0,00 | 129 | 0,00 | 129 | 0,00 |
| 27 | 173 | 173 | 0,00 | 173 | 0,00 | 173 | 0,00 | 173 | 0,00 | 173 | 0,00 | 173 | 0,00 | 173 | 0,00 |
| 28 | 163 | 163 | 0,00 | 163 | 0,00 | 163 | 0,00 | 163 | 0,00 | 163 | 0,00 | 163 | 0,00 | 163 | 0,00 |

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 29 | 94 | 92 | 2,13 | 70 | 25,53 | 70 | 25,53 | 70 | 25,53 | 70 | 25,53 | 43 | 54,26 | 43 | 54,26 |
| 30 | 180 | 180 | 0,00 | 180 | 0,00 | 180 | 0,00 | 180 | 0,00 | 180 | 0,00 | 180 | 0,00 | 180 | 0,00 |
| 31 | 82 | 81 | 1,22 | 71 | 13,41 | 81 | 1,22 | 81 | 1,22 | 81 | 1,22 | 81 | 1,22 | 81 | 1,22 |
| 32 | 78 | 78 | 0,00 | 78 | 0,00 | 78 | 0,00 | 78 | 0,00 | 78 | 0,00 | 77 | 1,28 | 77 | 1,28 |
| 33 | 70 | 70 | 0,00 | 70 | 0,00 | 70 | 0,00 | 70 | 0,00 | 70 | 0,00 | 70 | 0,00 | 70 | 0,00 |
| 34 | 286 | 286 | 0,00 | 283 | 1,05 | 283 | 1,05 | 283 | 1,05 | 283 | 1,05 | 209 | 26,92 | 209 | 26,92 |
| 35 | 289 | 289 | 0,00 | 287 | 0,69 | 287 | 0,69 | 287 | 0,69 | 287 | 0,69 | 283 | 2,08 | 283 | 2,08 |
| 36 | 277 | 272 | 1,81 | 272 | 1,81 | 272 | 1,81 | 272 | 1,81 | 272 | 1,81 | 248 | 10,47 | 248 | 10,47 |
| 37 | 554 | 554 | 0,00 | 539 | 2,71 | 539 | 2,71 | 539 | 2,71 | 539 | 2,71 | 332 | 40,07 | 332 | 40,07 |
| 38 | 182 | 180 | 1,10 | 180 | 1,10 | 180 | 1,10 | 180 | 1,10 | 180 | 1,10 | 104 | 42,86 | 104 | 42,86 |
| 39 | 251 | 149 | 40,64 | 97 | 61,35 | 84 | 66,53 | 84 | 66,53 | 84 | 66,53 | 58 | 76,89 | 58 | 76,89 |
| 40 | 154 | 154 | 0,00 | 154 | 0,00 | 154 | 0,00 | 154 | 0,00 | 154 | 0,00 | 17 | 88,96 | 0 | 100,00 |
| 41 | 200 | 200 | 0,00 | 200 | 0,00 | 200 | 0,00 | 200 | 0,00 | 200 | 0,00 | 145 | 27,50 | 145 | 27,50 |

En este vídeo, no se producen fallos de importancia hasta el *frame* 6. Únicamente los contornos 29 y 39 pierde gran cantidad de los puntos, pero ninguno más sufre pérdidas de importancia. Es a partir de éste *frame* cuando muchos de los contornos pierden una cantidad considerable de los puntos. Por tanto, es cuando debería realizarse la reasignación.

**Tabla 3 .** Número de puntos en los contornos en el vídeo 3.

| contorno | frame 0 | frame 1 | diferencia | frame 2 | diferencia | frame 3 | diferencia | frame 4 | diferencia | frame 5 | diferencia | frame 6 | diferencia | frame 7 | diferencia |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 83 | 83 | 0,00 | 44 | 46,99 | 44 | 46,99 | 44 | 46,99 | 37 | 55,42 | 37 | 55,42 | 37 | 55,42 |
| 1 | 209 | 209 | 0,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 |
| 2 | 190 | 188 | 1,05 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 |
| 3 | 203 | 203 | 0,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 |
| 4 | 184 | 184 | 0,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 |
| 5 | 140 | 140 | 0,00 | 4 | 97,14 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 |
| 6 | 223 | 223 | 0,00 | 30 | 86,55 | 30 | 86,55 | 30 | 86,55 | 18 | 91,93 | 0 | 100,00 | 0 | 100,00 |
| 7 | 260 | 260 | 0,00 | 22 | 91,54 | 22 | 91,54 | 22 | 91,54 | 4 | 98,46 | 0 | 100,00 | 0 | 100,00 |
| 8 | 288 | 288 | 0,00 | 10 | 96,53 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 |
| 9 | 302 | 302 | 0,00 | 23 | 92,38 | 23 | 92,38 | 23 | 92,38 | 23 | 92,38 | 23 | 92,38 | 23 | 92,38 |
| 10 | 183 | 183 | 0,00 | 6 | 96,72 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 |
| 11 | 118 | 118 | 0,00 | 19 | 83,90 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 |
| 12 | 119 | 119 | 0,00 | 21 | 82,35 | 21 | 82,35 | 21 | 82,35 | 17 | 85,71 | 0 | 100,00 | 0 | 100,00 |
| 13 | 176 | 176 | 0,0 | 0 | 100, | 0 | 100, | 0 | 100, | 0 | 100,0 | 0 | 100,0 | 0 | 100,0 |

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 0 | | 00 | | 00 | | 00 | | 0 | | 0 | | 0 |
| 14 | 156 | 156 | 0,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 |
| 15 | 235 | 235 | 0,00 | 233 | 0,85 | 233 | 0,85 | 233 | 0,85 | 200 | 14,89 | 200 | 14,89 | 143 | 39,15 |
| 16 | 246 | 240 | 2,44 | 35 | 85,77 | 34 | 86,18 | 34 | 86,18 | 30 | 87,80 | 25 | 89,84 | 25 | 89,84 |
| 17 | 219 | 219 | 0,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 |
| 18 | 195 | 141 | 27,69 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 |
| 19 | 240 | 240 | 0,00 | 84 | 65,00 | 84 | 65,00 | 84 | 65,00 | 72 | 70,00 | 60 | 75,00 | 56 | 76,67 |
| 20 | 210 | 210 | 0,00 | 208 | 0,95 | 208 | 0,95 | 208 | 0,95 | 197 | 6,19 | 197 | 6,19 | 147 | 30,00 |
| 21 | 199 | 198 | 0,50 | 56 | 71,86 | 56 | 71,86 | 56 | 71,86 | 47 | 76,38 | 44 | 77,89 | 43 | 78,39 |
| 22 | 169 | 169 | 0,00 | 69 | 59,17 | 69 | 59,17 | 69 | 59,17 | 66 | 60,95 | 66 | 60,95 | 66 | 60,95 |
| 23 | 91 | 91 | 0,00 | 91 | 0,00 | 91 | 0,00 | 91 | 0,00 | 91 | 0,00 | 91 | 0,00 | 91 | 0,00 |
| 24 | 180 | 180 | 0,00 | 103 | 42,78 | 103 | 42,78 | 103 | 42,78 | 101 | 43,89 | 48 | 73,33 | 48 | 73,33 |
| 25 | 166 | 164 | 1,20 | 145 | 12,65 | 143 | 13,86 | 143 | 13,86 | 133 | 19,88 | 133 | 19,88 | 133 | 19,88 |
| 26 | 154 | 139 | 9,74 | 131 | 14,94 | 131 | 14,94 | 131 | 14,94 | 129 | 16,23 | 129 | 16,23 | 96 | 37,66 |
| 27 | 134 | 134 | 0,00 | 4 | 97,01 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 |
| 28 | 214 | 19 | 91,12 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 |
| 29 | 191 | 191 | 0,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 |
| 30 | 200 | 200 | 0,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 |
| 31 | 230 | 199 | 13,48 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 | 0 | 100,00 |

En este vídeo, es en el segundo *frame* cuando casi todos los contornos pierden la mayoría de puntos que contienen, por tanto es donde debería realizarse la reasignación.
Observando los resultados obtenidos en los tres vídeos en los que se ha ejecutado el código, la pérdida crítica de puntos ocurre en el *frame* 4 para el primer vídeo, en el 6 en el segundo, y en el 2 en el tercero. Por tanto, hallando el número medio del *frame* en el que se debe realizar la reasignación de los puntos, se llega a la conclusión de que ésta debe hacerse en el *frame* 4.

**ANEXO 2. Puntos en los contornos tras la reasignación**

Al inicio del vídeo, el número de puntos asignados a los vectores de los contornos es el obtenido tras la ejecución del algoritmo de extracción de líneas. Como se ha explicado en el apartado X, se producen fallos en el seguimiento de estos puntos con el algoritmo de Lucas-Kanade, y éstos puntos mal seguidos se eliminan del contorno en el que estaban. Por tanto, debe realizarse una reasignación de los puntos de los contornos cada cierto tiempo, para asegurar el correcto seguimiento de las proyecciones de las rectas. Ésta reasignación se realiza cada cuatro *frames*, momento en el que se considera que se ha perdido un número considerable de puntos, y cuya justificación se realiza en el anexo 2.
En las siguientes tablas se muestran los puntos que contiene cada contorno en el inicio del vídeo y en el *frame* 3, justo antes de la reasignación.

**Tabla 1.** Puntos en los contornos del vídeo 1 antes de la reasignación.

| **Contorno** | **Puntos en el *frame* 0** | **Puntos en el *frame* 3** |
|---|---|---|
| 0 | 142 | 142 |
| 1 | 165 | 144 |
| 2 | 145 | 115 |
| 3 | 140 | 117 |
| 4 | 144 | 140 |
| 5 | 248 | 245 |
| 6 | 579 | 579 |
| 7 | 108 | 108 |
| 8 | 104 | 92 |
| 9 | 104 | 104 |
| 10 | 109 | 109 |
| 11 | 201 | 180 |
| 12 | 206 | 205 |
| 13 | 59 | 59 |
| 14 | 137 | 136 |
| 15 | 237 | 163 |
| 16 | 191 | 191 |
| 17 | 143 | 139 |
| 18 | 252 | 252 |
| 19 | 196 | 196 |
| 20 | 131 | 129 |
| 21 | 245 | 169 |
| 22 | 220 | 220 |
| 23 | 354 | 339 |
| 24 | 319 | 316 |
| 25 | 174 | 174 |

**Tabla 2.** Puntos en los contornos del vídeo 2 antes de la reasignación.

| Contorno | Puntos en el *frame* 0 | Puntos en el *frame* 3 |
|---|---|---|
| 0 | 215 | 215 |
| 1 | 200 | 198 |
| 2 | 170 | 168 |
| 3 | 161 | 161 |
| 4 | 121 | 106 |
| 5 | 105 | 90 |
| 6 | 115 | 115 |
| 7 | 241 | 225 |
| 8 | 151 | 138 |
| 9 | 112 | 112 |
| 10 | 134 | 134 |
| 11 | 107 | 106 |
| 12 | 218 | 205 |
| 13 | 214 | 214 |
| 14 | 163 | 163 |
| 15 | 245 | 199 |
| 16 | 175 | 175 |
| 17 | 142 | 142 |
| 18 | 154 | 154 |
| 19 | 95 | 95 |
| 20 | 77 | 76 |
| 21 | 306 | 300 |
| 22 | 164 | 157 |
| 23 | 62 | 62 |
| 24 | 105 | 105 |
| 25 | 175 | 175 |
| 26 | 129 | 129 |
| 27 | 173 | 173 |
| 28 | 163 | 163 |
| 29 | 94 | 70 |
| 30 | 180 | 180 |
| 31 | 82 | 81 |
| 32 | 78 | 78 |
| 33 | 70 | 70 |
| 34 | 286 | 283 |
| 35 | 289 | 287 |
| 36 | 277 | 272 |
| 37 | 554 | 539 |
| 38 | 182 | 180 |
| 39 | 251 | 84 |
| 40 | 154 | 154 |

| | | |
|---|---|---|
| 41 | 200 | 200 |

**Tabla 3.** Puntos en los contornos del vídeo 3 antes de la reasignación.

| **Contorno** | **Puntos en el *frame* 0** | **Puntos en el *frame* 3** |
|---|---|---|
| 0 | 83 | 44 |
| 1 | 209 | 0 |
| 2 | 190 | 0 |
| 3 | 203 | 0 |
| 4 | 184 | 0 |
| 5 | 140 | 4 |
| 6 | 223 | 30 |
| 7 | 260 | 22 |
| 8 | 288 | 10 |
| 9 | 302 | 23 |
| 10 | 183 | 6 |
| 11 | 118 | 19 |
| 12 | 119 | 21 |
| 13 | 176 | 0 |
| 14 | 156 | 0 |
| 15 | 235 | 233 |
| 16 | 246 | 34 |
| 17 | 219 | 0 |
| 18 | 195 | 0 |
| 19 | 240 | 84 |
| 20 | 210 | 208 |
| 21 | 199 | 56 |
| 22 | 169 | 69 |
| 23 | 91 | 91 |
| 24 | 180 | 103 |
| 25 | 166 | 143 |
| 26 | 154 | 131 |
| 27 | 134 | 4 |
| 28 | 214 | 2 |
| 29 | 191 | 0 |
| 30 | 200 | 0 |
| 31 | 230 | 0 |

Como se puede observar, en los tres primeros *frames* se han perdido puntos respecto a los que existían en el inicio de los vídeos. Cabe destacar la pérdida de puntos del vídeo 3, en el que algunos de los contornos han perdido la totalidad de los puntos, y en el que en el anexo 1 se puede ver como es en el *frame* 2 en el que se realiza un fallo de seguimiento para la gran mayoría de los puntos.

En las siguientes tablas se recoge el número de puntos en el *frame* 3 y en el *frame* 4, momento en el que se realiza la reasignación.

**Tabla 4.** Puntos en los contornos del vídeo 1 tras la reasignación.

| **Contorno** | **Puntos en el *frame* 3** | **Puntos en el *frame* 4** |
|---|---|---|
| 0 | 142 | 260 |
| 1 | 144 | 669 |
| 2 | 115 | 584 |
| 3 | 117 | 312 |
| 4 | 140 | 523 |
| 5 | 245 | 361 |
| 6 | 579 | 635 |
| 7 | 108 | 375 |
| 8 | 92 | 523 |
| 9 | 104 | 286 |
| 10 | 109 | 302 |
| 11 | 180 | 605 |
| 12 | 205 | 323 |
| 13 | 59 | 191 |
| 14 | 136 | 175 |
| 15 | 163 | 498 |
| 16 | 191 | 432 |
| 17 | 139 | 477 |
| 18 | 252 | 517 |
| 19 | 196 | 382 |
| 20 | 129 | 321 |
| 21 | 169 | 483 |
| 22 | 220 | 312 |
| 23 | 339 | 524 |
| 24 | 316 | 357 |
| 25 | 174 | 413 |

**Tabla 5.** Puntos en los contornos del vídeo 2 tras la reasignación.

| **Contorno** | **Puntos en el *frame* 3** | **Puntos en el *frame* 4** |
|---|---|---|
| 0 | 215 | 94 |
| 1 | 198 | 133 |
| 2 | 168 | 244 |
| 3 | 161 | 367 |
| 4 | 106 | 155 |
| 5 | 90 | 211 |
| 6 | 115 | 276 |

| | | |
|---|---|---|
| 7 | 225 | 251 |
| 8 | 138 | 224 |
| 9 | 112 | 244 |
| 10 | 134 | 294 |
| 11 | 106 | 245 |
| 12 | 205 | 324 |
| 13 | 214 | 383 |
| 14 | 163 | 357 |
| 15 | 199 | 421 |
| 16 | 175 | 90 |
| 17 | 142 | 234 |
| 18 | 154 | 194 |
| 19 | 95 | 212 |
| 20 | 76 | 283 |
| 21 | 300 | 291 |
| 22 | 157 | 196 |
| 23 | 62 | 184 |
| 24 | 105 | 255 |
| 25 | 175 | 379 |
| 26 | 129 | 168 |
| 27 | 173 | 162 |
| 28 | 163 | 171 |
| 29 | 70 | 343 |
| 30 | 180 | 115 |
| 31 | 81 | 94 |
| 32 | 78 | 106 |
| 33 | 70 | 218 |
| 34 | 283 | 359 |
| 35 | 287 | 304 |
| 36 | 272 | 362 |
| 37 | 539 | 488 |
| 38 | 180 | 123 |
| 39 | 84 | 162 |
| 40 | 154 | 105 |
| 41 | 200 | 114 |

**Tabla 6.** Puntos en los contornos del vídeo 3 tras la reasignación.

| **Contorno** | **Puntos en el *frame* 3** | **Puntos en el *frame* 4** |
|---|---|---|
| 0 | 44 | 176 |
| 1 | 0 | 0 |
| 2 | 0 | 0 |
| 3 | 0 | 0 |
| 4 | 0 | 0 |

| | | |
|---|---|---|
| 5 | 0 | 0 |
| 6 | 30 | 151 |
| 7 | 22 | 86 |
| 8 | 0 | 0 |
| 9 | 23 | 108 |
| 10 | 0 | 0 |
| 11 | 0 | 0 |
| 12 | 21 | 104 |
| 13 | 0 | 0 |
| 14 | 0 | 0 |
| 15 | 233 | 112 |
| 16 | 34 | 206 |
| 17 | 0 | 0 |
| 18 | 0 | 0 |
| 19 | 84 | 134 |
| 20 | 208 | 175 |
| 21 | 56 | 219 |
| 22 | 69 | 183 |
| 23 | 91 | 124 |
| 24 | 103 | 120 |
| 25 | 143 | 202 |
| 26 | 131 | 201 |
| 27 | 0 | 0 |
| 28 | 0 | 0 |
| 29 | 0 | 0 |
| 30 | 0 | 0 |
| 31 | 0 | 0 |

En esta última tabla se puede ver cómo en el vídeo 3 sólo se ha realizado la reasignación de los puntos de los contornos que contienen más de 20 puntos, y que por tanto se considera que no se ha perdido la línea.

Podemos observar que en la mayoría de casos el número de puntos tras la reasignación es superior a los que anteriormente contenía el vector. Esto es producido porque en las reasignaciones se guardan todos los puntos de la imagen que son bordes, y que coinciden con la proyección de la recta. Como bien se observa en el apartado XX, además de los puntos del contorno que forma la línea, también se guardan puntos que son bordes y que se ajustan al plano de proyección. Es por esto que el número de puntos de los contornos es superior tras la reasignación.

En otras ocasiones el número de puntos tras la reasignación es inferior. Esto es producido por fallos de seguimiento de los puntos perdidos entre el *frame* anterior a la reasignación y el posterior.

## ANEXO 3. CÓDIGO C++

A continuación se va a exponer el código utilizado en éste trabajo, explicando paso por paso cómo se han realizado las tareas necesarias para el funcionamiento del algoritmo.
Para no extender innecesariamente este anexo, sólo se va a explicar la parte del código en la que carga el vídeo y ejecuta el seguimiento de las rectas. Previamente, se ha aplicado el algoritmo de obtención de líneas sobre el primer *frame* del vídeo y se han guardado los puntos que las forman en *puntosLineImages*, un vector que en cada una de sus celdas contiene el vector de puntos de cada línea. Por tanto, *puntosLineImages* es un vector de vectores de puntos que van a ser seguidos.

Antes de cargar el vídeo se introducen varios contadores: *cont, contframes* y *NumFramesReasignacion* se utilizan para guardar los *frames* y ejecutar las acciones requeridas, y *PuntosMin* es el mínimo de puntos que debe tener un contorno para considerar que la línea no se ha perdido.

```
int cont = 0;
int contframes = 0;
int NumFramesReasignacion = 4;
int PuntosMin = 20;

for (;;)
{
        cap >> image;

        if (!image.data)
        {
                cout << "error al cargar el frame";
                exit(1);
        }

        img_next = image.clone();


        cvtColor(img_prev, grayA, CV_RGB2GRAY);
        cvtColor(img_next, grayB, CV_RGB2GRAY);

        if (contframes < NumFramesReasignacion)
        {
                contframes++;
        }
```

Las siguientes acciones se ejecutan cuando *contframes* es inferior a *NumFramesReasginacion*. Se realiza el seguimiento de los puntos que forman las líneas, mediante la función *calcOpticalFlowPyrL,* que es la contenida en las librerías *OpenCV* que ejecuta el algoritmo de seguimiento de puntos de Lucas Kanade piramidal. La posición de los puntos en el siguiente frame se guarda en *PuntosASeguir.* Posteriormente, se pinta una línea que une cada punto con su posición almacenada en *PuntosASeguir* con la orden *line*. Esta línea sólo se pinta si se quiere observar el correcto seguimiento de los puntos.
Las acciones descritas sólo se ejecutan si el contorno tiene un número de puntos superior a *PuntosMin*. De no ser así, se considera que se ha perdido esa línea y no se sigue en los *frames* posteriores.

```
Mat PlotImg;
PlotImg = img_prev.clone();
vector<vector<Point2f>> puntosASeguir(puntosLineImages.size());
vector<Point2f> vectorzeros(0);
for (int i = 0; i < puntosLineImages.size(); i++)
{
        if (puntosLineImages.at(i).size() > PuntosMin)
        {
                calcOpticalFlowPyrLK(grayA, grayB, puntosLineImages.at(i),
                puntosASeguir.at(i), status, error, Size(winsize, winsize),
                maxlvl);


                for (int j = 0; j < puntosLineImages.at(i).size(); j++)
                {
                        line(PlotImg, puntosLineImages.at(i).at(j),
                        puntosASeguir.at(i).at(j), Scalar(0, 255, 0), 2);
                }
        }
        else
        {
                puntosASeguir.at(i) = vectorzeros;
        }

}
```

Tras el seguimiento de los puntos, se procede a eliminar los puntos cuyo seguimiento no se ha realizado correctamente. Se establece un umbral *distmax*, de manera que los puntos cuya posición en el siguiente *frame* se aleja más de 20 píxeles de su posición actual se considera que ha habido un fallo en su seguimiento y se eliminan del vector de puntos que los contiene.

```
int distmax = 20;
for (int i = 0; i < puntosASeguir.size(); i++)
{
        vector<Point2f> PuntosBuenos;

        for (int j = 0; j < puntosASeguir.at(i).size(); j++)
        {

                float Xdist = puntosASeguir.at(i).at(j).x -
                        puntosLineImages.at(i).at(j).x;
                float Ydist = puntosASeguir.at(i).at(j).y -
                        puntosLineImages.at(i).at(j).y;
                float dist = sqrt(pow(Xdist, 2) + pow(Ydist, 2));

                if (dist < distmax)
                {

                        if ((puntosASeguir.at(i).at(j).x <
                        imMask.cols) && (puntosASeguir.at(i).at(j).x
                        >0) && (puntosASeguir.at(i).at(j).y <
                        imMask.rows) && (puntosASeguir.at(i).at(j).y
                        > 0) && (imMask.at<unsigned
                        char>(Point2f(puntosASeguir.at(i).at(j).x,
                        puntosASeguir.at(i).at(j).y)) == 0))
                                {

                        PuntosBuenos.push_back(puntosASeguir.at(i).at(j));

                                }

                        }


                }

        puntosASeguir.at(i) = PuntosBuenos;
}
```

Por último, tan sólo queda obtener la línea que forma cada contorno. Para ello se utiliza la función *ransacExtractionFromBundary*, que extrae el plano de proyección de la recta, y lo define mediante su vector normal *ransacDataOut.nMax*, y *plotParametricCurve* para pintar ésta proyección. Para ello, los puntos que forman la recta deben estar referenciados desde el centro de la imagen y en modo matriz, y no en modo vector y desde la esquina superior izquierda como se guardan por defecto. Por ello se realiza este cambio, y las coordenadas de los puntos almacenados en *puntosASeguir* pasan a la matriz *xHatBound*.

```
        Vec3b color(0, 0, 255);

        for (int k = 0; k < puntosASeguir.size(); k++)
        {
                if (puntosASeguir.at(k).size() > PuntosMin)
                {
                        Mat xHatBound = Mat_<float>(2, puntosASeguir.at(k).size());
                        vector<Point2f> SavedPoints;

                        for (int kk = 0; kk < puntosASeguir.at(k).size(); kk++)
                        {
                                xHatBound.at<float>(0, kk) =
                                (float)puntosASeguir.at(k).at(kk).x - u0;
                                xHatBound.at<float>(1, kk) =
                                (float)puntosASeguir.at(k).at(kk).y - v0;
                        }

                RansacDataOut ransacDataOut =
                ransacExtractionFromBoundary(xHatBound, 10, calib, modelClass, rThreshold);

                plotParametricCurve(PlotImg, ransacDataOut.nMax, calib, modelClass,
                u0, v0, color);

                }
        }
```

Esto es la secuencia de funciones que se ejecuta mientras no se realiza la reasignación de los puntos. Ésta se ejecuta cuando el numero de *frames* es *NumFramesReasignacion*.
Lo primer es volver a asignar el valor 0 al contador *contframes*, para que se vuelva a contar desde el principio una vez hecha la reasignación. Posteriormente se pasa la imagen a blanco y negro y se le aplica el algoritmo de *Canny* para hallar los bordes de la imagen. Tras este algoritmo, se obtiene una imagen binaria con los bordes en blanco sobre un fondo negro. Por tanto, se guardan todos estos puntos blancos en el vector *PuntosBordes*, que contiene todos los puntos de la imagen que han sido considerados como bordes.
Estos puntos se van a comparar con las líneas, que recordemos que están definidas desde el el centro de la imagen y en modo matriz. Por tanto, las coordenadas del vector *PuntosBordes* también se pasan a modo matriz referenciada desde el centro de la imagen, pasando a la matriz *BordesHat.*

```
else
        {
        contframes = 0;
        Mat Bordes, bluredImg;
        blur(grayA, bluredImg, Size(3, 3));
        Canny(bluredImg, Bordes, lowThreshold, lowThreshold*ratio, kernel_size);

        vector<Point2f> PuntosBordes;

        for (int i = 0; i < Bordes.cols; i++)
        {
                for (int j = 0; j < Bordes.rows; j++)
                {
                        unsigned char ColorPixel = Bordes.at<unsigned
                        char>(Point(i, j));
                        if (ColorPixel == 255)
                        {
                                Point2f PixelBlanco;
                                PixelBlanco.x = i;
                                PixelBlanco.y = j;
                                PuntosBordes.push_back(PixelBlanco);
                        }

                }
        }


        Mat_<float> BordesHat = Mat_<float>(2, PuntosBordes.size());

        for (int kk = 0; kk<PuntosBordes.size(); kk++)
        {
                BordesHat.at<float>(0, kk) = PuntosBordes.at(kk).x - u0;
                BordesHat.at<float>(1, kk) = PuntosBordes.at(kk).y - v0;
        }
```

Ahora se procede a comparar la posición de *BordesHat* con la de las líneas. Para ello se pasan las coordenadas de *puntosLineImages*, si contiene un número de puntos superior al mínimo, a matriz referenciada desde el centro de la imagen, y se hallan las proyecciones de las líneas que forman esos contornos. Después, se ejecuta la orden *getdAlg*, una función que dada una matriz de puntos y el vector normal del plano que contiene la proyección de la línea, guarda en el vector *aAlgArray* la distancia de cada uno de los puntos de la matriz a la recta. Por último, si dicha distancia es inferior al umbral *PointDist*, ese punto se ajusta a la línea y pasa a formar parte del vector de puntos del contorno que forma esa línea.

```
for (int k = 0; k < puntosLineImages.size(); k++)
{
        if (puntosLineImages.at(k).size() > PuntosMin)
        {
                Mat xHatBound = Mat_<float>(2, puntosLineImages.at(k).size());

                for (int kk = 0; kk < puntosLineImages.at(k).size(); kk++)
                        {
                        xHatBound.at<float>(0, kk) = puntosLineImages.at(k).at(kk).x -
                        u0;
                        xHatBound.at<float>(1, kk) = puntosLineImages.at(k).at(kk).y -
                        v0;
                        }

                RansacDataOut ransacDataOut = ransacExtractionFromBoundary(xHatBound, 10,
                calib,  modelClass, rThreshold);

                Mat_<float> linea = ransacDataOut.nMax;

                vector<float> dAlgArray = getdAlg(BordesHat, linea, calib, modelClass);


                float PointDist = 1;

                vector<Point2f> PuntosCercanos;

                for (int i = 0; i < dAlgArray.size(); i++)
                {

                        if (dAlgArray.at(i) < PointDist)

                        {

                        Point2f puntoxy(BordesHat.at<float>(0, i) + u0,
                        BordesHat.at<float>(1, i)               + v0);

                        PuntosCercanos.push_back(puntoxy);

                        }
                }
        puntosLineImages.at(k) = PuntosCercanos;

        }
}
```

**BIBLIOGRAFIA**


[1] K. Valavanis, *Advances in unmanned aerial vehicles: state of the art and the road to autonomy*, University of South Florida, Tampa, Florida, 2007.

[2] S.M. Adams, C.J. Friedland, *A survey of unmanned aerial vehicle (UAV) usage for imagery collection in disaster research and management,* Universidad de Louisiana, 2011.

[3] S. Waharte, N. Trigoni, *Supporting Search and Rescue Operations with UAVs*, publicado en Emerging Security Technologies (EST), 2010 International Conference.

[4] S. Rathinam, P. Almeida, Z. Kim, S. Jackson, A. Tinka, W. Grossman, R. Sengupta, *Autonomous Searching and Tracking of a River using an UAV*, American Control Conference, New York, 2007.

[5] F. Mancimi, M. Dubbini, M. Gattelli, F. Steechi, S. Fabbri, G. Gabbianelli, *Using Unmanned Aerial Vehicles (uav) for High-Resolution Reconstruction of Topography: The Structure from Motion Approach on Coastal Environment,* publicado en remote sensing, pages 6880-6898, 2013

[6] F. Clapuyt, V. Vanacker, K. Van Oost, *Reproducibility of UAV-based earth topography reconstructions based on Structure-from-Motion algorithms*, publicado en Geomorphology Volume 260, pages 4–15, 1 May 2016.

[7] H. Saari, I. Pellikka, L. Pesonen, S. Tuominen, J. Heikkila, C. Holmlund, J. Makynen, K. Ojala, T. Antila, *Unmanned Aerial Vehicle (UAV) operated spectral camera system for forest and agriculture applications*, Remote Sensing for Agriculture, Ecosystems, and Hydrology XIII, Prague, Czech Republic, 2011.

[8] Y, Hui, C. Xhiping, X. Shanjia, W. Shisong, *An unmanned air vehicle (UAV) GPS location and navigation system*, publicado en Microwave and Millimeter Wave Technology Proceedings, pages 472-475, Beijing, 1998.

[9] S. Lange, N. Sunderhauf, P. Protzel*, A vision based onboard approach for landing and position control of an autonomous multirotor UAV in GPS-denied environments*, publicado en Advanced Robotics, 2009. ICAR 2009. International Conference

[10] R. Mur-Artal, J.M.M. Montiel, J.D. Tardós, ORB-SLAM: A Versatile and Accurate Monocular SLAM System, IEEE Transactions on Robotics, vol. 31, no. 5, pages. 1147-1163, October 2015.

[11] D. Gálvez-López, M. Salas, J. D. Tardós, J.M.M. Montiel, *Real-time monocular object SLAM*, publicado en Sciencedirect Robotics and Autonomous Systems, Volume 75, Part B, , pages 435–449, January 2016

[12] bibliotecas *OpenCV* , www.opencv.org.

[13] A. Escalera, J.M. Armingol, F. García, D. Martín, A.H. Al-Kaff, curso online *Introducción a la visión por computador: desarrollo de aplicaciones con OpenCV* de la universidad Carlos III de Madrid, en la plataforma edx courses, 2016.

[14] J. Bermudez-Cameo, G. Lopez-Nicolas , J.J. Guerrero, *Automatic Line Extraction in Uncalibrated Omnidirectional Cameras with Revolution Symmetry*, International Journal of Computer Vision, 114(1), pages 16-37, 2015

[15] J. Canny, *A Computational Approach To Edge Detection*, IEEE Trans. Pattern Analysis and Machine Intelligence, Volume: PAMI-8, Issue: 6, pages 679 - 698, 1986.

[16] M. A. Fischler, R. C. Bolles, *Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography*, Communications of the ACM, 1981.

[17] B. K. P. Horn, B. G. Schunk, *Determining optical flow*, published in Artificial Intelligence Volume 17, Issues 1–3, pages 185-203, August 1981,

[18] B.D. Lucas, T. Kanade, *An iterative image registration technique with an application to stereo vision*, publicado en Proceedings DARPA Image Understanding Workshop, pages 121-130, April 1981.

[19] www.cs.otago.az.nz, Computer Vision Research Group , Department of Computer Science, University of Otago Dunedin, New Zealand.

[20] C. Harris, M. Stephens*, A combined corner and edge detector*, publicado en Proceedings of the 4th Alvey Vision Conference, pages 147–151, 1994.

[21] J. Shi, Tomasi, *Good features to track*, Computer Vision and Pattern Recognition, 1994. IEEE Computer Society Conference.

[22] J. Y. Bouguet, *Pyramidal Implementation of the Affine Lucas Kanade Feature Tracker Description of the algorithm*, Microprocessor Research Labs, Intel Corporation.

[23] J. C. Bazin, C. Demonceaux, P. Vasseur, I.S. Kweon, *Motion estimation by decoupling rotation and translation in catadioptric vision*, publicado en Computer Vision and ImageUnderstanding Volume 114, Issue 2, pages 254–273, February 2010.