Escuela Superior de Ingenieros de Sevilla

Ingeniería de Sistemas y Automática

Proyecto fin de carrera

SISTEMA PARA LA COMPENSACIÓN DEL MOVIMIENTO EN SECUENCIAS DE IMÁGENES AÉREAS

Titulación: Ingeniero de Telecomunicación

Autor:
Fernando Caballero Benítez

Tutores:
Aníbal Ollero Baturone
Joaquín Ferruz Molero

A mis padres, sin su ayuda no habría podido llegar hasta aquí.

A Vicky, Gallego y Fran, por apoyarme cuando más lo he necesitado.

A Rafa, Marcos, Luis, Víctor, Alqui, Luis Merino, Fernando, Migue y resto de compañeros del laboratorio, por las veces que me habéis ayudado y por las risas que nos hemos echado.

A todos, gracias.

Índice

Capítulo 1

Introducción

1.1. Proyecto COMETS.

El acrónimo COMETS procede de "Real-time coordination and control of multiple heterogeneous unmanned aerial vehicles". El objetivo principal del proyecto es diseñar e implementar un sistema de control distribuido para la detección y monitorización usando vehículos aéreos heterogéneos. Los vehículos usados serán dos helicópteros y un zeppelín.

Con el fin de conseguir este objetivo, en el proyecto se diseñará e implementará una nueva arquitectura de control, desarrollando nuevas técnicas e integrando un sistema distribuido de sensores y tratamiento digital de imágenes en tiempo real.

Para poder validar estos conceptos y sistemas, COMETS demostrará el funcionamiento en la detección de incendios forestales. Esta es una difícil misión donde la cooperación entre vehículos se hace perfectamente visible.

El proyecto que se describe en este documento se enmarca dentro de la parte de procesamiento digital de imágenes a la que anteriormente se hacía referencia en COMETS.

El análisis de imágenes tomadas desde helicóptero nos puede facilitar datos que las tomas fijas en tierra no pueden tomar, ya que en vuelo podemos obtener mayor variedad de ángulos y movimiento.

En términos generales, el tratamiento digital de imágenes para su posterior estudio y extracción de datos se facilita notablemente si la escena que se observa se mantiene fija, ya que las referencias de posición permanecen inalterables a lo largo del proceso de análisis. Cuando dicho proceso se realiza sobre secuencias de imágenes aéreas tenemos, sin embargo, el inconveniente de que se introducen en ellas efectos debidos a las turbulencias y a las propias vibraciones del helicóptero, lo que les produce fuertes perturbaciones. Las escenas, que se suponen fijas, contarán con movimiento de forma constante, a pesar de que el helicóptero se mantenga en hovering.

Nos encontramos por tanto con las enormes posibilidades que nos puede ofrecer el análisis digital de imágenes tomadas desde helicóptero, pero con el problema intrínseco de las vibraciones en las mismas. Una posible mejora podría ser la corrección de la posición de las imágenes de forma digital, en un ordenador. De esta forma podríamos usar las grabaciones tomadas desde el aire, sin el inconveniente que ello conlleva.

Así pues, en este proyecto se pretende el análisis de la compensación del movimiento en secuencias de imágenes tomadas desde helicóptero. El método que se propone consiste en, para cada imagen que nos llega desde la cámara aérea, deducir cuál

ha sido el movimiento que se ha producido respecto a la anterior y deshacerlo, de tal modo que la escena que se está analizando permanezca fija en el plano de la imagen.

1.2. Introducción a la memoria del proyecto.

En un primer lugar analizaremos de forma más exhaustiva el problema al que nos enfrentamos y cuales son las características de este, analizándose los datos de los que dispondremos y los datos a los que pretendemos llegar u obtener.

Conocido el problema con exactitud, definiremos un algoritmo que nos permitirá la compensación de secuencias de imágenes aéreas. Será necesario en este punto describir en partes se divide el proceso que hemos diseñado, especificando el funcionamiento de cada una de estas secciones con profundidad.

Posteriormente, veremos que el procesamiento necesario para poder llevar a cabo el proceso es muy elevado, de modo que se definirán mejoras que nos permitirán reducir el tiempo de procesamiento.

Por último, se pretende señalar algunas posibles ampliaciones sobre el proyecto. Algunas de estas ampliaciones puede que tenga el estudio teórico ya diseñado en la memoria, otras en cambio es necesario desarrollarlas desde el principio.

Capítulo 2

Especificación del problema

2.1. Introducción.

En este punto se pretende analizar de forma más exhaustiva cuales son las condiciones de las que partimos y que datos nos serán necesarios para poder dar solución al problema que se nos plantea.

Los datos de partida sobre los que nos apoyaremos serán los siguientes:

- Secuencia de imágenes sobre la que compensar el movimiento.
- Datos de la imagen a compensar.

Por otro lado, los datos o servicios que debe ofrecer el programa son:

- La imagen con la compensación de movimiento ya hecha.
- La matriz de homografía, cuyas características se detallarán más adelante, que es la herramienta que se utilizará para realizar dicha compensación.

El método que vamos a utilizar para poder deducir el movimiento que se ha producido de una imagen a la siguiente está basado en la búsqueda de regiones de la imagen anterior en la imagen actual. A este método se le denomina "Seguimiento de Ventanas" y está ya implementado, de modo que no será objetivo de este proyecto su desarrollo, pero si será necesario referenciar cuales son las características de este y en que se fundamenta.

En sucesivos puntos veremos cuales con las características del método de seguimiento, así como los datos de partida y los datos ofrecidos por el sistema de compensación que pretendemos diseñar.

2.2. Método de seguimiento de ventanas.

Junto con la imagen, los datos obtenidos de la aplicación de este método a la secuencia serán unos de los más relevantes del proceso, ya que nos aportará información a cerca de cual ha sido el movimiento que se ha producido de una imagen a al siguiente.

Podemos definir el seguimiento de ventanas como un proceso de búsqueda de determinadas regiones en una imagen. Dichas regiones (ventanas), fueron seleccionadas en la imagen anterior y al buscarlas en la actual, la intención es obtener el movimiento

que han sufrido, para poder deducir de él la alteración que produjo la diferencia entre las dos imágenes y así, más adelante, poder realizar la operación inversa.

El proceso de seguimiento de ventanas se puede dividir en las siguientes partes:

- *Selección de ventanas:* consiste en señalar las regiones de la imagen que se buscarán en la siguiente. El criterio de selección de ventanas es importante, ya que en función de éste la probabilidad de encontrar la región posteriormente será mayor o menor.

  Básicamente, el criterio utilizado será la elección de ventanas cuyo contraste sea lo suficientemente elevado como para poder ser fácilmente identificables, y por tanto, ser encontradas. Este criterio se enfrenta con el problema de asociar ventanas a bordes y esquinas que pueden desaparecer de una imagen a la siguiente. Posteriormente se verán métodos para evitar este tipo de circunstancias.

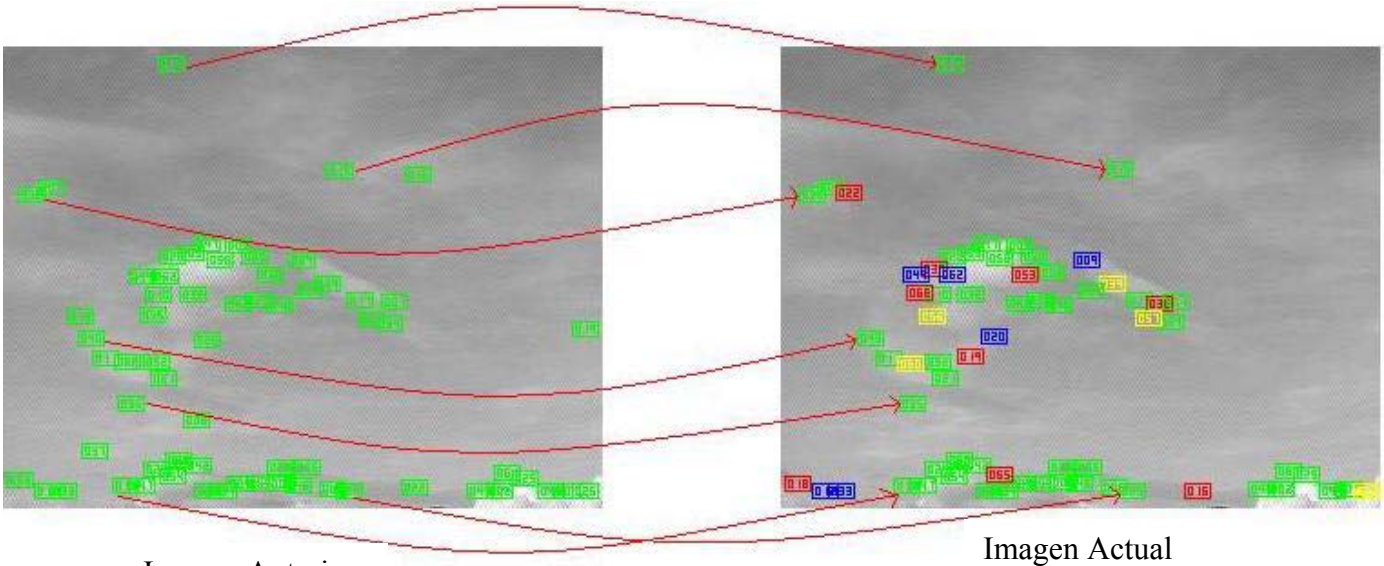


Imagen Anterior

Imagen Actual

Figura 1.

Es necesario tener en cuenta que no se deben seleccionar regiones cercanas al borde de la imagen, ya que la probabilidad de desaparecer en la siguiente imagen es muy elevada debido al movimiento.

- *Búsqueda de regiones en la siguiente imagen:* Conocidas cuáles son las ventanas, debemos buscar las regiones correspondientes en la imagen actual. El proceso de búsqueda está basado en la correlación de la región en el entorno de la posición original de la ventana, es decir, se correla la región en torno a su centro, seleccionándose la zona donde la correlación es mayor. Si esa

correlación es mayor que una cota determinada se escogerá la zona como nueva posición.

Al mismo tiempo se hace una búsqueda basada en clusters. Gracias a estos clusters podemos garantizar la distancia relativa entre ventanas en el seguimiento, dando una mayor robustez al seguimiento de regiones.

Dada la posición de una ventana en una imagen y la posición de esa misma ventana en la imagen siguiente, denominaremos correspondencia a dicha pareja de posiciones. Una correspondencia nos informa acerca de dónde se encuentra una ventana en la imagen actual y dónde se encontraba en la imagen anterior.

- *Selección de nuevas regiones:* Durante el proceso de búsqueda se pueden producir pérdidas de ventanas (aquellas que no son encontradas), es por ello que se deben seleccionar nuevas regiones que reemplacen a las que se han perdido.

Como se dijo anteriormente, este método no forma parte de los objetivos del proyecto y está ya implementado, de modo que nuestro programa hará uso de el en cada una de las iteraciones del mismo.

Es necesario tener en cuenta que el método de seguimiento de ventanas trabaja solo con imágenes en blanco y negro. Esto implica que, si la secuencia de imágenes a compensar es en color, será necesario pasarlas a blanco y negro para poder utilizar esta herramienta en ella y así obtener la información deseada.

Existen diferentes procesos para pasar de una imagen en color a otra en blanco y negro. El problema que se nos plantea es que no podemos invertir demasiado tiempo en este proceso ya que no es más que una adaptación de los datos a un formato concreto. El método que se ha decidido seguir consiste en el promediado del valor de intensidad del píxel de la siguiente manera:

Intensidad = (1/3) *(Intensidad rojo + Intensidad verde + Intensidad azul)

Este procedimiento no es el más adecuado debido a que el espacio RGB no es homogéneo, y por tanto el factor de escala del promediado no es el mismo para los tres planos. Sin embargo, se optado por su uso porque como característica positiva encontramos que la conversión a blanco y negro se hace de forma mucho más sencilla y rápida.

El método de seguimiento de ventanas posee una serie de parámetros a los que será necesario dar valores. Algunos de los parámetros más importantes podrían ser los siguientes:

- *Nivel del autovalor:* Este parámetro nos permite configurar la sensibilidad de la selección de las ventanas. Cuando el nivel del autovalor es demasiado pequeño se escogen ventanas de contraste bajo, las cuales son muy difíciles de encontrar en la siguiente imagen. Por el contrario, si el nivel del autovalor es lo suficientemente elevado se consigue escoger regiones de contraste alto, aunque, recordemos, esto puede llegar a ser perjudicial, ya que se podrían centrar todas las regiones en torno a los bordes y las esquinas, las cuales, en determinadas situaciones, pueden generar problemas en el seguimiento.

- *Número máximo de regiones:* Este parámetro indica el número máximo de ventanas a seleccionar y buscar en la siguiente imagen. Su importancia es grande. El número de regiones debe ser lo suficientemente alto como para tener una idea adecuada del movimiento en la imagen completa. Un valor típico para imágenes de 300x200 suele ser 180. Valores excesivamente elevados de este parámetro pueden generar un procesamiento extremadamente largo y por tanto desperdicio de tiempo.

- *Tolerancia en la búsqueda:* Este parámetro representa a dos parámetros reales del proceso de seguimiento de ventanas. Éstos son "error_ini" y "error_fin", e indican la tolerancia a la hora de decidir si la búsqueda de una región ha sido correcta o no. El procedimiento sería el siguiente: La región seleccionada en la imagen anterior se correla en la imagen actual en un entorno de la posición original. Sólo los puntos que poseen un error de autovalor inferior a "error_ini" se estudian y, de éstos, sólo los que tienen un error de correlación inferior a "error_fin".

2.3. Datos de la imagen a compensar.

Los datos de la imagen que nos serán necesarios son los relativos a las características de las mimas. Algunos de los más importantes serían los siguientes:

- Ancho de la imagen.

- Alto de la imagen.

- Región de interés a compensar

Como resulta lógico, estos datos los necesitamos para ajustar la búsqueda a la zona de la imagen deseada. Además necesitamos saber el tipo de imagen para hacer la compensación.

La región de interés nos va a permitir indicar la zona de la imagen que queremos compensar, es decir, conocido cuál ha sido el movimiento que se ha producido en la imagen podemos compensarlo para la imagen en su totalidad, o sólo para la zona que nos resulte de interés, como por ejemplo la zona de estudio.

2.4. Servicios del proceso de compensación de movimiento.

Los servicios aportados por el proceso serán básicamente la imagen con la compensación de movimiento aplicada y la matriz de homografía que se ha utilizado para generar dicha compensación. El concepto de matriz de homografía lo veremos más a delante, pero por ahora bastará decir que esta matriz contiene la información acerca del movimiento producido de una imagen a la siguiente. Uno de las tareas de nuestro programa deberá ser hallar dicha matriz.

En términos generales bastará con la imagen compensada, pero en algunos casos, por requerimientos del usuario nos puede hacer falta solo la matriz de homografía. Es por esto por lo que el proceso de compensación de movimiento debe aportar esta matriz.

Capítulo 3

Descripción del algoritmo de compensación de movimiento

3.1. Introducción.

El algoritmo que se ha utilizado para la estabilización de imágenes está basado en la información obtenida del proceso de seguimiento aplicado a la secuencia de imágenes. Este algoritmo se puede dividir en las siguientes partes:

- *Obtención de la matriz de homografía:* La homografía es una matriz que nos permite expresar el movimiento que se ha producido de una imagen a la siguiente. En esta sección del algoritmo se obtiene dicha matriz, aplicando diversos procesos numéricos para garantizar que el resultado es el correcto

- *Aplicación de la matriz de homografía sobre la imagen para compensar el movimiento:* Conocida la matriz de homografía podemos aplicarla a la imagen para generar la compensación del movimiento producido

El diagrama de estados del proceso de compensación de movimiento lo podemos encontrar en la Figura 2.

En las secciones que siguen se analizarán cada una de estas partes por separado.

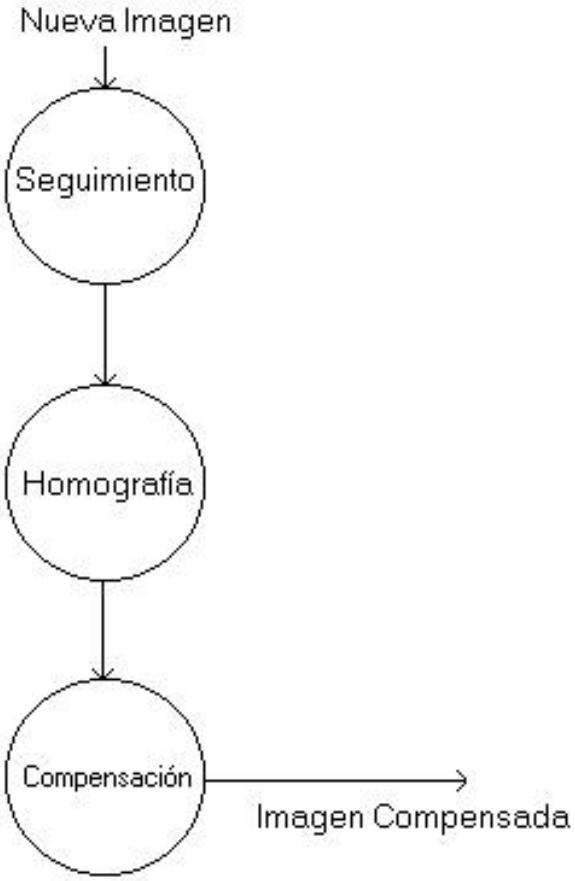


Figura 2.

En la Figura 2 podemos ver por tanto tres partes bien diferenciadas en el proceso de compensación de imágenes: el seguimiento de la imagen mediante el método de seguimiento de ventanas, la obtención del movimiento mediante la matriz de homografía y la aplicación de la compensación sobre la imagen.

Como se dijo anteriormente, en este proyecto no se estudiar a fondo el seguimiento de ventanas por nos ser el objetivo principal del mismo, centrándonos más en la obtención de la matriz de homografía y en la compensación de la imagen conocida dicha matriz.

3.2. Obtención de la matriz de homografía.

3.2.1. Concepto de homografía.

La matriz de homografía no es más que una transformación plana sobre una imagen. Esta transformación se puede caracterizar por aplicar una matriz de dimensión 3x3. De esta forma una homografía cualquiera se puede expresar por:

$$H = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix}$$

El coeficiente igual a "1" es debido a que una matriz de homografía es igual a ella misma multiplicada por una constante, de modo que se suele tomar como convenio poner dicho coeficiente a "1" para simplificar los cálculos.

Esta transformación se aplica sobre la posición (soy) de cada píxel de la siguiente manera:

$$\begin{bmatrix} x'k \\ y'k \\ k \end{bmatrix} = H * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Vemos cómo la transformación no es lineal, ya que está multiplicada por el coeficiente k, obteniéndose ésta a partir de las ecuaciones.

Si se realiza un análisis de la influencia de cada uno de los coeficientes en la matriz de homografía podemos deducir que:

- *Coeficientes a, b, d, e:* En ellos se guarda la rotación que se ha producido de una imagen a otra, respecto a la esquina superior izquierda de la misma. De esta manera, si se ha producido un giro de P grados, los valores de *a, b, d* y *e* serán los siguientes:

$$a = \cos(P) \qquad b = -\text{sen}(P) \qquad d = \text{sen}(P) \qquad e = \cos(P)$$

- *Coeficientes c, f:* Estos dos coeficientes indican un desplazamiento de la imagen, indicando *c* el desplazamiento en el eje x y *f* el desplazamiento en el eje y.

- *Coeficientes g, h:* Estos son los coeficientes que generan las no linealidades de la transformación y nos permiten realizar escalados y deformaciones sobre la imagen a la que se la aplicamos.

Generalmente se suele clasificar la matriz de homografía de la forma siguiente:

- *Matriz de homografía:* Este es el tipo de matriz de homografía anteriormente especificado. Para calcularla son necesarios, como mínimo, cuatro correspondencias entre una imagen y otra. Dichas correspondencias no pueden estar en la misma recta, deben estar suficientemente distribuidas por la imagen para obtener un buen resultado. A mayor número de correspondencias, más fiable es la transformación obtenida.

- *Matriz de homografía afín:* Esta matriz de homografía es igual a la anterior, salvando que en este caso los coeficientes *g* y *h* son nulos. Esto genera que la transformación deje de ser no lineal, siendo mucho más fácil de resolver. En este caso el número de correspondencias mínimo para la obtención de dicha transformación es 3.

Como veremos más adelante, la no linealidad de la transformación se puede resolver, o al menos minimizar, mediante una linealización a tramos de la misma. Las no linealidades de la matriz de homografía convencional se pueden ver en el siguiente ejemplo, donde se muestra, en la Figura 3, la evolución de la posición x e y al aplicarle la siguiente matriz de homografía no lineal:

$$H = \begin{bmatrix} 0.555570 & -0.831469 & 1 \\ 0.831469 & 0.555570 & 3 \\ 0 & 0.02 & 1 \end{bmatrix}$$

Y después la misma evolución con la matriz de homografía afín correspondiente, en la Figura 4:

$$H = \begin{bmatrix} 0.555570 & -0.831469 & 1 \\ 0.831469 & 0.555570 & 3 \\ 0 & 0 & 1 \end{bmatrix}$$

Vemos cómo la evolución de las posiciones del nuevo eje x' e y' no es lineal

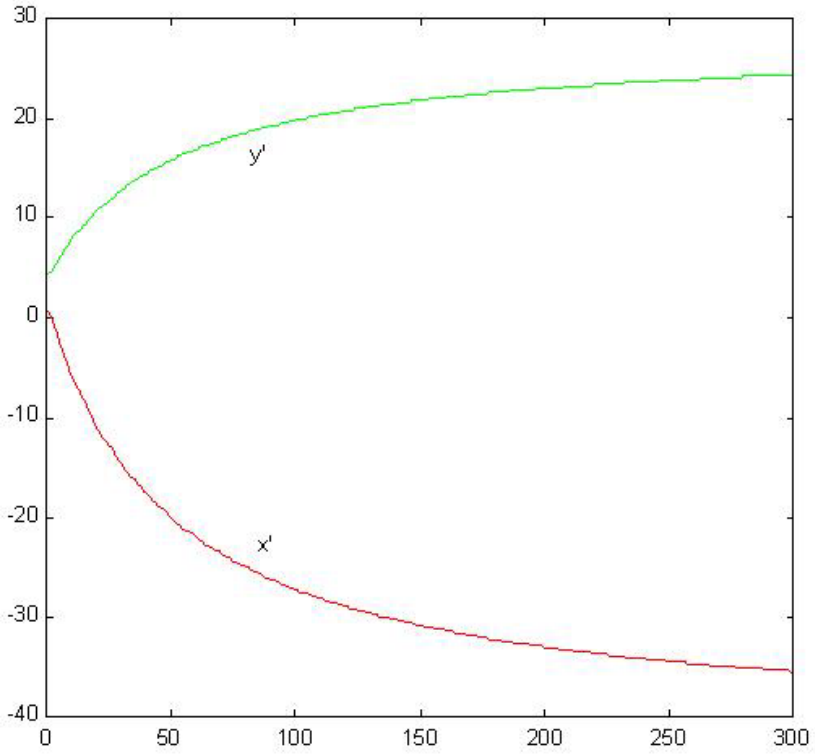


Figura 3.

Pero si la matriz es afín, los resultados son completamente lineales

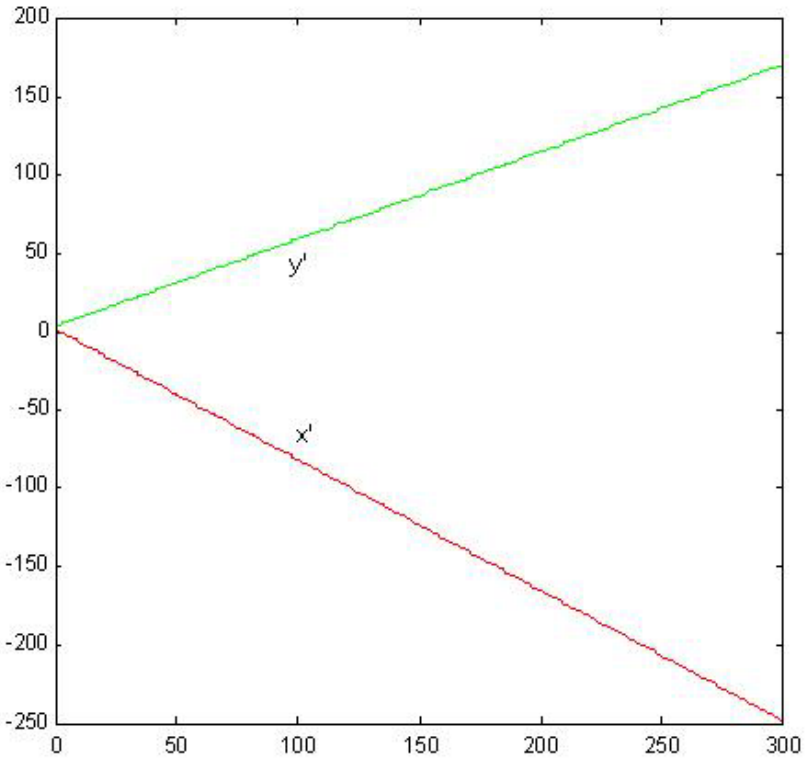


Figura 4.

El uso de la matriz de homografía para la compensación de imágenes se puede hacer gracias a la suposición de que el terreno es plano. Esta suposición es valida en términos generales dado que, a la distancia a la que se suele encontrar el helicóptero, la superficie enfocada por la cámara se puede aproximar por un plano. A pesar de todo, es conveniente situarse siempre a la altura adecuada para que esta premisa sea lo más cierta posible.

3.2.2. Secciones del algoritmo para obtener la matriz de homografía.

Se pretende en este proyecto obtener la matriz de homografía a partir de la información aportada por la posición de las correspondencias de una a otra imagen. Para conseguirlo será necesario en primer lugar filtrar los datos acerca de las posiciones de las correspondencias de modo que podamos eliminar errores. Posteriormente, se aplicarán técnicas numéricas para la obtención de la matriz de homografía que mejor se ajuste a los datos. El diagrama de estados del algoritmo que obtiene la matriz de homografía lo encontramos en la Figura 5.

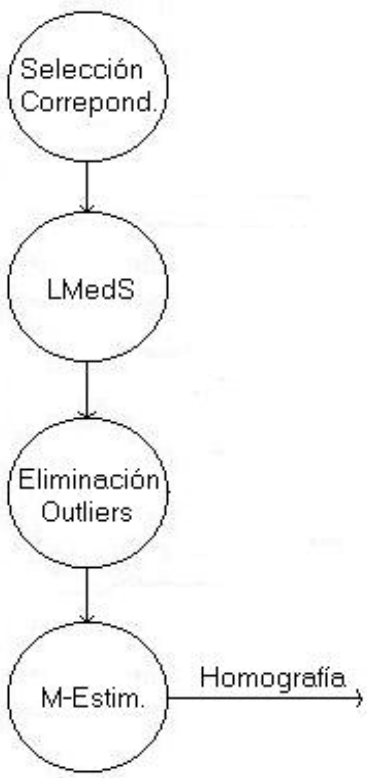


Figura 5.

Para poder filtrar los datos de entrada se ha decidido aplicar el algoritmo de mínima mediana de los mínimos cuadrados (LMedS). Este algoritmo nos va a permitir obtener una desviación típica de los datos, con la que seremos capaces de establecer una cota máxima de error la cual nos permitirá la detección de datos incorrectos.

El cálculo definitivo de la matriz de homografía lo haremos mediante un algoritmo de M-Estimaciones, que servirá para establecer con gran precisión el valor de la matriz de homografía a partir de los valores ya filtrados. Este algoritmo también nos permitirá detectar errores en los datos de las correspondencias.

De forma sencilla podemos decir que este proceso esta encargado de obtener la matriz que permita invertir el movimiento producido en una imagen

.

3.2.2.1. Método de LMedS.

El método de la mínima mediana de los mínimos cuadrados estima los valores de la matriz de homografía resolviendo el siguiente problema no lineal de minimización:

$$\min_{i} \operatorname{med} r_i^2$$

Esto significa que se debe averiguar el valor mínimo de las medianas de los residuos al cuadrado, aplicado a todo el conjunto de datos. Éste es un método muy robusto para la detección de correspondencias erróneas y, por tanto, para la detección de outliers en general.

El valor de la mínima mediana lo podemos calcular mediante la búsqueda de la mejor estimación de la matriz de homografía en el espacio de los datos. Debido a que analizar todas las posibilidades existentes sería demasiado costoso, es necesario aplicar algún método aleatorio de selección de datos.

El algoritmo de LMedS se puede describir de la siguiente manera. Partimos de un conjunto de n correspondencias $m_i=(x_i,y_i)^T$:

- En primer lugar se debe utilizar algún algoritmo de Montecarlo para la selección de m conjuntos de p elementos del espacio de datos.

- Para cada uno de los conjuntos de p elementos, indexados mediante j, calcularemos la matriz de homografía H_j que mejor se ajusta a éstos.

- Para cada H_j debemos calcular la mediana de los residuos al cuadrado, M_j, respecto a los valores que indica la correspondencia.

  $$M_j = \operatorname*{med}_{i=1....n} r_i^2(H_j,m_i)$$

  Para calcular el valor de r_i podemos usar diferentes técnicas. En nuestro caso usaremos la distancia euclídea entre la posición calculada con la matriz H_j y el valor de la correspondencia.

- Se guarda el valor de H_j que posee el menor M_j entre todos los M_j.

La cuestión ahora es cómo determinar el número de conjuntos m que garantice la obtención de una buena H_j. Asumiendo que entre todo el conjunto de correspondencias existe un conjunto E de outliers, la probabilidad de que al menos uno de los m conjuntos sea bueno será:

$$Pb = 1 - \left[1 - (1 - E)^p\right]^m$$

Imponiendo que el valor de Pb debe ser cercano a uno obtenemos que:

$$m = \frac{\log(1 - Pb)}{\log(1 - (1 - E)^p)}$$

La eficiencia del método de LMedS es bastante pobre ante ruido Gaussiano. Dicha eficiencia es definida como el cociente entre la varianza menos alcanzable por los parámetros estimados y la varianza dada por el método. De forma experimental se llega a que la varianza de los datos se puede expresar como sigue:

$$\sigma = 1.4826\left(1 + \frac{5}{n - p}\right)\sqrt{M_J}$$

Conocida la varianza del espacio de datos, podemos utilizarla para detectar cuáles son outliers, de modo que si la distancia r_i^2 es mayor que $2.5\sigma^2$ el dato será un outlier, siendo un dato correcto en caso contrario.

Tal y como se adelantaba anteriormente, al aplicar el algoritmo de los LMedS nos encontramos con problemas a la hora de usarlo sobre conjuntos de datos que estén muy cerca en el espacio, ya que en este caso el modelo que usamos para definir la transformación puede no converger o ajustarse de forma inadecuada a los datos. Para evitar esta circunstancia, en [2] Zhang aconseja el uso de lo que denomina Buckets.

El uso de las rejillas no es más que la división del espacio de datos en un conjunto de cuadrados o cubos. Con esto pretendemos que cuando se tome uno de los datos de dentro de uno de esos cubos, no se vuelva a tomar ninguno otro más del mismo. Así nos estamos asegurando que los datos que se toman en cada iteración del LMedS están lo suficientemente alejados como para no generar problemas de convergencia al obtener el modelo que se ajuste a dichos datos.

Si tenemos una imagen, una posible forma de distribuir las rejillas por la misma es la que se muestra en la Figura 6.

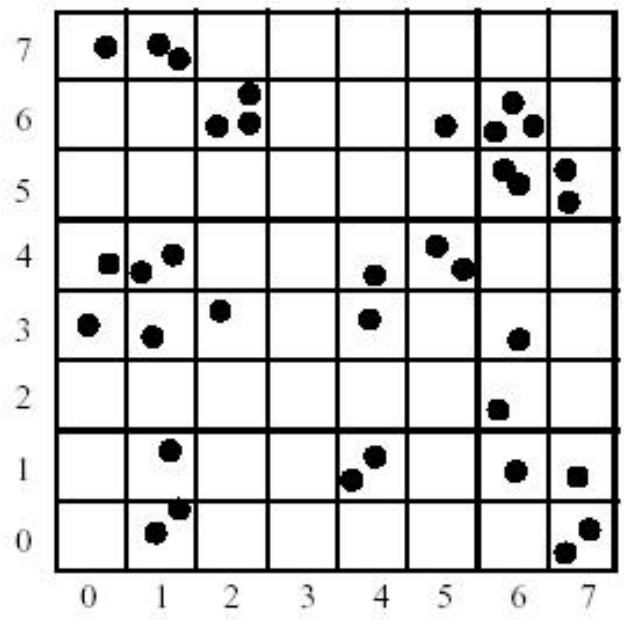


Figura 6.

A la vista de la Figura 6, queda claro que es necesario que el número de buckets y su distribución sea adecuada a las circunstancias, de modo que se adapten correctamente a los datos de los que se dispone. Más adelante veremos la solución que tomada.

3.2.2.2. Implementación del método de LMedS.

En nuestro caso, la implementación que se ha llevado a cabo ha sido ligeramente diferente con la intención de dar mayor robustez a la detección de los outliers.

Como principal diferencia tendremos que al algoritmo de LMedS no accederán todas las correspondencias correctas, sino sólo aquéllas que tengan determinadas características. La mejora consiste en suponer que una correspondencia que fue outlier en la iteración anterior, probablemente seguirá siéndolo en la iteración presente, de modo que es conveniente que se tenga en cuenta a la hora de obtener la desviación típica de los datos.

Igualmente, tampoco se tendrán en cuenta las correspondencias nuevas, ya que no tenemos ningún tipo de información acerca de la fiabilidad de dicha correspondencia.

Con lo anteriormente especificado podemos llegar a una estructura selectiva que nos va a permitir separar entre tres tipos de correspondencias:

- *Correspondencias incorrectas:* Son aquellas correspondencias que el proceso de seguimiento de ventanas indica que no son correctas.

- *Correspondencias adecuadas para LMedS:* Éstas son correspondencias correctas y que además fueron correctas en la iteración pasada, no siendo outlier.

- *Correspondencias adecuadas para M-Estimaciones:* Estas correspondencias son correctas, pero, o bien son nuevas, o bien fueron outliers en la iteración anterior.

A la hora de determinar si una correspondencia pertenece a un tipo u otro deberemos seguir la siguiente lógica:

- Si la correspondencia es incorrecta ➔ Correspondencia incorrecta.

- Si la correspondencia es correcta y nueva ➔ Correspondencia adecuada para M-Estimaciones.

- Si la correspondencia es correcta y fue outlier en la iteración pasada ➔ Correspondencia adecuada para M-Estimaciones.

- Si la correspondencia es correcta y no fue outlier en la iteración anterior ➔ Correspondencia adecuada para LMedS.

Conocida cuál va a ser la lógica de selección de las correspondencias, en el algoritmo de LMedS, sólo usaremos aquéllas que sean adecuadas para el mismo. Sobre este subconjunto de datos aplicaremos el algoritmo.

En nuestro caso, el modelo que se aplicará sobre los datos será el de la matriz de homografía. Debido a que las modificaciones que se pueden producir sobre la imagen no tienen porqué ser lineales, ya que se pueden producir pequeñas deformaciones sobre la misma, el modelo de homografía que aplicaremos será la completa.

Al aplicar el modelo de matriz de homografía completo, el número de puntos mínimo necesarios para hallar dicha matriz será 4, ya que la matriz posee 8 incógnitas y cada correspondencia añade 2 ecuaciones al sistema.

Es necesario tener en cuenta que las correspondencias que se seleccionen para cada subconjunto de 4 no deben estar alineadas ni demasiado cerca unas de otras. Veamos a continuación los posibles problemas que aparecen en la agrupación de las correspondencias:

- *Correspondencias alineadas:* En este caso, tal y como se dijo anteriormente, el sistema resultante de este conjunto de correspondencias, donde al menos dos están alineadas, puede ser irresoluble o generar un modelo de la transformación incoherente.

- *Correspondencias muy cercanas entre si:* Esta situación genera que la matriz de homografía resultante sea correcta, pero que solo tenga en cuenta la transformación de ese conjunto de correspondencias. Esto producirá un residuo muy pequeño en las correspondencias cercanas a las que han generado la homografía y un error muy grande en las correspondencias muy alejadas. Como consecuencia se puede caer en el error de tomar como outliers las correspondencias más alejadas. Por tanto, debemos evitar en todo momento escoger conjuntos de puntos muy cercanos entre si.

Para evitar los problemas anteriores debemos dividir la imagen en buckets de tamaño adecuado a las circunstancias. Existen varias formas de hacerlo, pero entre ellas podemos destacar básicamente dos:

- *División de la imagen en buckets de tamaño muy reducido:* En este caso, cada vez que se toma una correspondencia de una rejilla es necesario inhibir las rejillas que se encuentran alrededor de éste, de modo que la próxima vez que cojamos una correspondencia no esté cerca de la tomada anteriormente.

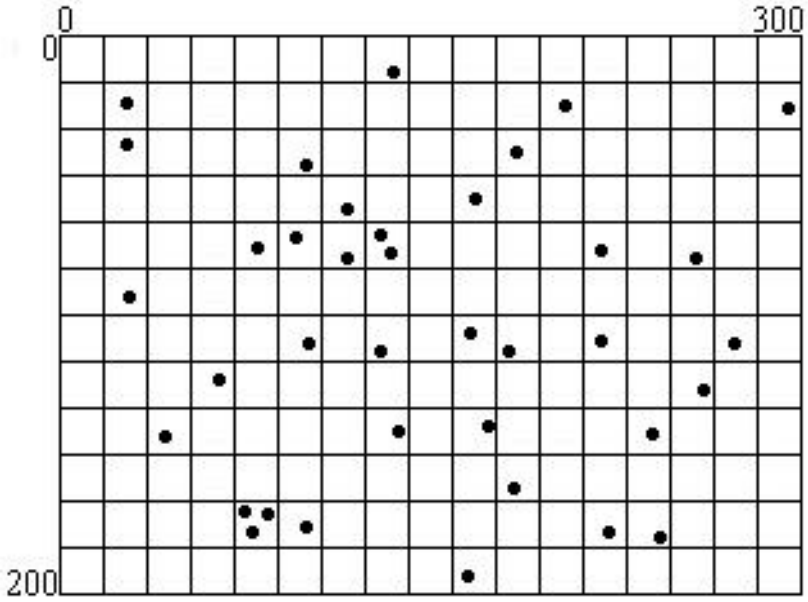


Figura 7.

- *División de la imagen en buckets de gran tamaño:* Ahora no tenemos por qué preocuparnos de anular las rejillas que se encuentran alrededor de la rejilla del

cuál hemos tomado la correspondencia. Este caso es mucho más simple y supone menos procesamiento.

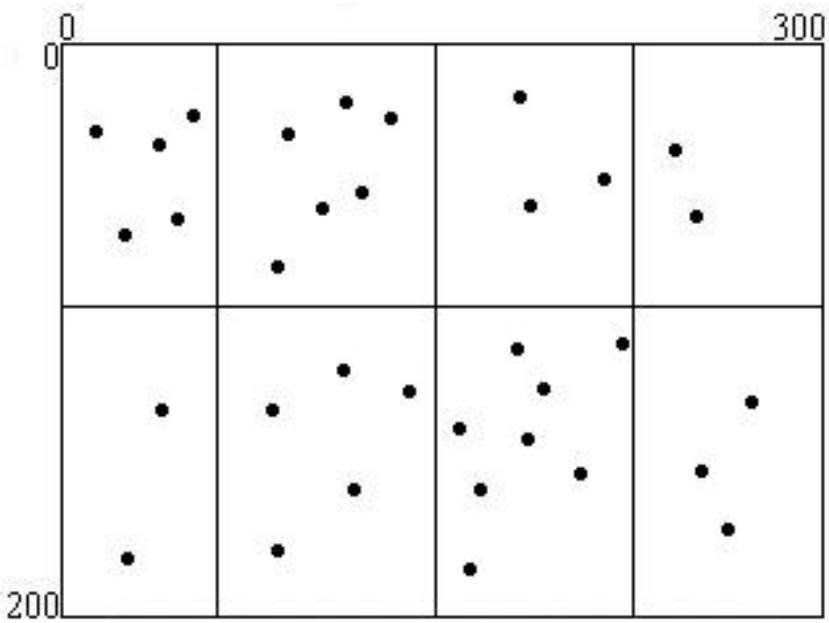


Figura 8.

Nosotros usaremos el esquema de la Figura 8, por ser el que mejores resultados nos proporciona, ya que lo único que debemos hacer es anular la rejilla del cual hemos cogido una correspondencia.

En cada iteración se tomará una correspondencia y se anulará la rejilla que la contenga, de modo que llegará un momento en que no quede ningún bucket sin utilizar, en cuyo caso se resetearán las rejillas, pudiéndose volver a coger correspondencias de cada uno de ellos.

De gran importancia es la forma en la que elegimos las correspondencias de entre todas las demás. Como dijimos anteriormente, para la selección de correspondencias se usa un algoritmo aleatorio, concretamente, de Montecarlo. Los algoritmos aleatorizados son aquellos que en algún momento de su ejecución, utilizan la generación de un número aleatorio para tomar alguna decisión. En muchos casos, este comportamiento puede hacer a un algoritmo más eficiente. Los algoritmos aleatorizados se dividen en dos grandes familias, a saber:

- *Algoritmos tipo Las Vegas:* Estos algoritmos siempre resuelven correctamente el problema, sin embargo en ciertas ocasiones (con baja probabilidad de ocurrencia) pueden tardar un tiempo promedio significativamente superior a la media.

- *Algoritmos tipo Montecarlo:* Siempre son eficientes en cuanto a la cantidad de tiempo necesaria para encontrar la respuesta, sin embargo en algunos casos (con baja probabilidad de ocurrencia) pueden producir una respuesta incorrecta del problema.

Así, para la selección hemos usado un método a dos niveles, en primer lugar se elige de forma aleatoria, según distribución uniforme, la rejilla del cual se va a tomar la correspondencia y a continuación, también según distribución uniforme, se elige una correspondencia de entre todas las que posee la rejilla. Posteriormente se inhibe la rejilla para que no pueda volverse a utilizar hasta que no queden buckets disponibles. Procediendo de este modo tenemos garantizada la aleatoriedad en la elección de las correspondencias.

Por último veremos como solventar el problema de la resolución de sistemas de ecuaciones con mayor número de ecuaciones que incógnitas. Esta situación se nos puede dar si especificamos como número mínimo de puntos un valor superior a 4, es decir si tomamos grupos de correspondencias de más de 4 elementos.

Para la resolución de estos sistemas sobredeterminados tenemos que aplicar mínimos cuadrados sobre las ecuaciones. El proceso básico será el siguiente:

Partimos del sistema $Ax = B$ y multiplicamos a izquierda y derecha por A^T:

$$A^T Ax = A^T B$$

Tomamos $C = A^T A$ y $D = A^T B$, llegando al siguiente sistema cuadrado:

$$Cx = D$$

Como resultado nos queda un sistema de ecuaciones con el mismo número de ecuaciones que incógnitas, resoluble mediante diversos métodos.

Para hallar finalmente la solución del sistema de ecuaciones cuadrado, aplicaremos la descomposición LU a la matriz. Este tipo de descomposición nos permite expresar la matriz C como el producto de dos matrices, una triangula superior y otra triangular inferior, de modo que, suponiendo C matriz cuadrada de dimensión 4:

$$C = L * U = \begin{bmatrix} a & 0 & 0 & 0 \\ b & c & 0 & 0 \\ d & e & f & 0 \\ g & f & i & j \end{bmatrix} * \begin{bmatrix} k & l & m & n \\ 0 & p & q & r \\ 0 & 0 & s & t \\ 0 & 0 & 0 & u \end{bmatrix}$$

Conocidas las matrices L y U podemos resolver el sistema de ecuaciones de forma casi trivial:

Partimos de $Cx = D$

Sustituimos C = L*U

$$LUx = D$$

Sustituimos Ux = x'

$$Lx' = D$$

Resolvemos este sistema trivial y el resultado lo sustituimos en la ecuación de x', de la que obtenemos los valores de x.

Para poder hacer la descomposición LU de la matriz cuadrada usaremos funciones del "Numérical recipes in C".

3.2.2.3. Método de M-Estimaciones.

Éste es un método bastante usado para obtener estimaciones robustas. Consiste en iterar de forma continua en el calculo del modelo, de modo que en cada iteración se multiplique por pesos muy pequeños aquellas ecuaciones del sistema con errores residuales muy grandes, con esto conseguimos que el error residual vaya disminuyendo en cada una de las iteraciones del algoritmo. Cuando el error residual llega hasta una cota determinada, se detiene la ejecución y se devuelve el modelo al que se ha llegado después de las repeticiones.

Uno de los problemas principales a la hora de aplicar el método de M-Estimaciones es encontrar una función de penalización adecuada a la evolución del error en la determinación del modelo. Algunas posibles funciones de penalización podrían ser las que se proponen en la Figura 9.

type	$\rho(x)$	$\psi(x)$	$w(x)$
L_2	$x^2/2$	x	1
L_1	$\lvert x\rvert$	$\operatorname{sgn}(x)$	$\frac{1}{\lvert x\rvert}$
$L_1 - L_2$	$2(\sqrt{1+x^2/2}-1)$	$\frac{x}{\sqrt{1+x^2/2}}$	$\frac{1}{\sqrt{1+x^2/2}}$
L_p	$\frac{\lvert x\rvert^p}{p}$	$\operatorname{sgn}(x)\lvert x\rvert^{p-1}$	$\lvert x\rvert^{p-2}$
"Fair"	$c^2[\frac{\lvert x\rvert}{c} - \log(1+\frac{\lvert x\rvert}{c})]$	$\frac{x}{1+\lvert x\rvert/c}$	$\frac{1}{1+\lvert x\rvert/c}$
Huber $\begin{cases}\text{if } \lvert x\rvert \le k \\ \text{if } \lvert x\rvert \ge k\end{cases}$	$\begin{cases}x^2/2 \\ k(\lvert x\rvert - k/2)\end{cases}$	$\begin{cases}x \\ k\operatorname{sgn}(x)\end{cases}$	$\begin{cases}1 \\ k/\lvert x\rvert\end{cases}$
Cauchy	$\frac{c^2}{2}\log(1+(x/c)^2)$	$\frac{x}{1+(x/c)^2}$	$\frac{1}{1+(x/c)^2}$
German-McClure	$\frac{x^2/2}{1+x^2}$	$\frac{x}{(1+x^2)^2}$	$\frac{1}{(1+x^2)^2}$
Welsch	$\frac{c^2}{2}[1-\exp(-(x/c)^2)]$	$x\exp(-(x/c)^2)$	$\exp(-(x/c)^2))$
Tukey $\begin{cases}\text{if } \lvert x\rvert \le c \\ \text{if } \lvert x\rvert > c\end{cases}$	$\begin{cases}\frac{c^2}{6}(1-[1-(x/c)^2]^3) \\ (c^2/6)\end{cases}$	$\begin{cases}x[1-(x/c)^2]^2 \\ 0\end{cases}$	$\begin{cases}[1-(x/c)^2]^2 \\ 0\end{cases}$

Figura 9.

Es necesario destacar que el método de las M-Estimaciones no es válido cuando existen mínimos locales, ya que el método obtiene el modelo que mejor se ajusta al mínimo más cercano, si este mínimo es local, el resultado no será el óptimo. La forma más sencilla de asegurar que esto no ocurra es introduciendo datos que a priori tengan una alta probabilidad de ser correctos, con esto forzamos al sistema a que se coloque cercano al mínimo absoluto y por tanto se tienda al óptimo del modelo, si es que había mínimos locales.

Este problema se dejaba ver de forma bastante evidente en anteriores implementaciones de este algoritmo para la determinación de la matriz de homografía. En implementaciones previas se supuso que la matriz de homografía poseía 9 parámetros desconocidos, lo cual no es cierto, ya que uno de ellos se puede fijar a un valor constante. Como consecuencia, cuando se aplicaba el algoritmo de M-Estimaciones con una H de 9 parámetros, en algunas ocasiones, y sin motivo aparente, el algoritmo daba como resultado una matriz de homografía que era totalmente incoherente. La causa era que, al poseer un parámetro de más, el sistema poseía un mínimo local cercano al mínimo total, de modo que, en función de la certeza de los datos de entrada, el sistema se decantaba por un mínimo u otro, generando salidas incoherentes casi de forma arbitraria.

Observamos por tanto que, cuando existen mínimos locales en la resolución del modelo, la efectividad del M-Estimaciones depende directamente de los datos de entrada al mismo, de modo que si los datos son mayoritariamente correctos, se obtendrá una buena estimación del modelo, mientras que si los datos poseen una varianza un poco más elevada de lo normal, el modelo obtenido puede ser incorrecto.

En nuestro caso, aún existiendo mínimos locales, conseguimos buenos resultados gracias a que en pasos anteriores se eliminaron los outliers mediante el método de LMedS, lo cual nos garantiza que los datos de entrada al M-Estimador van a ser en su mayoría correctos y, en consecuencia, conllevarán un buen comportamiento del algoritmo.

3.2.2.4. Implementación del método de M-Estimaciones.

A la hora de implementar el método de M-Estimaciones hemos tenido en cuenta que los datos ya han sido filtrados por el algoritmo de LMedS y por tanto, la mayor parte de los datos son correctos.

La función de penalización de error que hemos utilizado es la siguiente:

$$\omega(x) = \frac{1}{1 + \frac{|x|}{C}}$$

En ella, el crecimiento del valor del error, *x*, implica una disminución en el valor de peso, *ω(x)*. Este peso multiplica a la ecuación de la que procede el error residual, por tanto, cuanto mayor sea el error residual, menos contará dicha ecuación en la obtención de la matriz de homografía mediante mínimos cuadrados.

Es necesario definir una condición de salida del bucle de iteraciones del algoritmo. En nuestro caso, hemos definido una condición basada en la evolución de los pesos de penalización y en la evolución de la matriz de homografía de una iteración a la siguiente. Para verificar la condición es necesario realizar lo siguiente:

- Obtener la suma del valor absoluto de la diferencia entre los elementos de la matriz de homografía actual y la anterior.

- Obtener la suma del valor absoluto de la diferencia entre los pesos de la iteración anterior y la actual.

- Finalmente, se suma el resultado de los dos puntos anteriores, obteniéndose un valor mayor que cero. La condición de salida del bucle es que dicho valor sea inferior a 10^{-3}. En caso contrario se seguirá iterando.

Debemos tener en cuenta que algunos de los datos que le lleguen al algoritmo pueden generar sistemas indeterminados, de modo que es necesario definir un número máximo de iteraciones. En nuestro caso hemos tasado dicho número máximo en 15.

Una vez obtenida la matriz de homografía definitiva, podemos llevar a cabo una nueva detección de outliers. Esta nueva detección la podemos hacer en función de la recién calculada matriz, para lo que tan sólo debemos transformar todas las correspondencias y obtener el error respecto al valor que debería dar. Aplicamos la misma desviación típica que en el LMedS y con ella detectamos los outliers.

3.3. Compensación de la Imagen.

Disponiendo ya de la matriz de homografía tenemos una idea bastante acertada acerca de cuál ha sido el movimiento de una a otra imagen. Conocido dicho movimiento, nuestro objetivo es mover la imagen actual de forma que lo contrarreste. En la Figura 10 podemos encontrar un ejemplo de los datos que nos aporta la matriz de homografía que disponemos. Básicamente, lo que haremos será aplicar la homografía sobre la imagen actual para poder compensar el movimiento.

En este punto veremos diferentes formas de hacer la compensación, la cuál será diferente en función del tipo de imagen al que se le aplique.

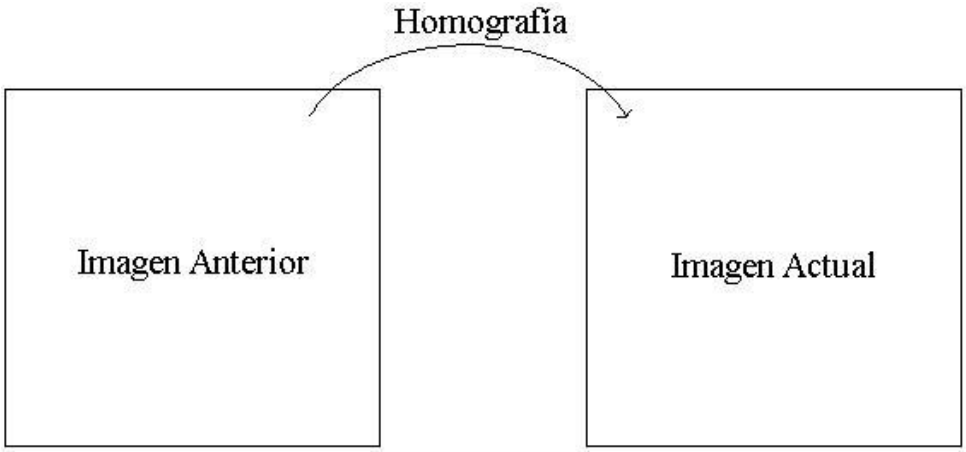


Figura 10.

3.3.1. Concepto de compensación.

Recordemos que la matriz de homografía nos indica a qué posición debe ir cada uno de los píxeles de la imagen anterior para que se pueda compensar el movimiento que los ha hecho ocupar posiciones diferentes a las actuales.

Suponiendo que la matriz de homografía no es afín, la forma de obtener la nueva posición de los píxeles sería:

$$H = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix}$$

$$x' = (x * H[0] + y * H[1] + H[2])/(x * H[6] + y * H[7] + H[8])$$
$$y' = (x * H[3] + y * H[4] + H[5])/(x * H[6] + y * H[7] + H[8])$$

Cada una de estas transformaciones nos generará una posición (x',y') a la que se desplazará el píxel (x,y). Pero dichas coordenadas, al haberse obtenido tras operaciones algebraicas, no tienen porqué ser dos números enteros, de modo que se plantea la cuestión de qué píxel es mejor asignar a dicha posición (Figura 11). Será necesario usar algún método para tomar esta decisión.

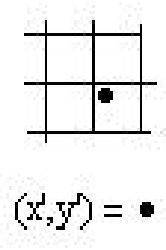


Figura 11.

Además de lo anterior, debemos tener en cuenta que la compensación de la imagen dependerá directamente del tipo de imagen a la que se lo estemos aplicando. Nosotros veremos tres tipos de imágenes:

- Imágenes en color.
- Imágenes en blanco y negro.
- Imágenes binarias.

Nos vemos obligados a definir diferentes modos de hacer la compensación, ya que los datos son diferentes para cada uno de estos tres tipos.

Sería interesante poder indicarle al proceso de compensación cuál es la zona de la imagen que se quiere compensar, ya que en algunas ocasiones solo nos interesa la parte sobre la cual estamos estudiando. Esta posibilidad podría dar mayor versatilidad al sistema y al mismo tiempo ahorraría tiempo, pues nos ahorraríamos procesar la transformación completa.

3.3.2. Compensación de imágenes binarias.

Las imágenes binarias están compuestas por píxeles cuyos únicos valores pueden ser 0 o 1, o lo que es lo mismo, "false" o "true". En estas imágenes no es conveniente aplicar el seguimiento de ventanas debido a que la imagen carece de los suficientes matices para que se puedan seguir las ventanas correctamente.

A pesar de todo, puede darse el caso de necesitar compensar el movimiento de imágenes binarias, usando como matriz de homografía aquella dada por una imagen real. De esta manera podemos estabilizar máscaras asociadas a imágenes o bien, especificando una matriz adecuada a nuestras necesidades, rotar y trasladar máscaras.

Por tanto, una vez conocida la matriz de homografía que se le pasa como parámetro al proceso, podemos aplicar la transformación a la posición de cada uno de los píxeles de la manera que se especificó anteriormente. El matiz que más diferencia este tipo de imágenes de las demás es la forma de asignar el valor del píxel una vez que se tiene la posición transformada. En concreto, el valor del píxel transformado vendrá dado por la siguiente regla lógica:

- Si el valor del píxel es diferente al valor de los cuatro píxeles vecinos ➔ El valor del píxel es el de los cuatro vecinos.

- Si el valor del píxel es, al menos, igual al valor de uno de sus vecinos ➔ El valor del píxel es el que inicialmente posee.

En la Figura 12 podemos encontrar una secuencia de dos imágenes binarias estabilizada mediante el algoritmo.

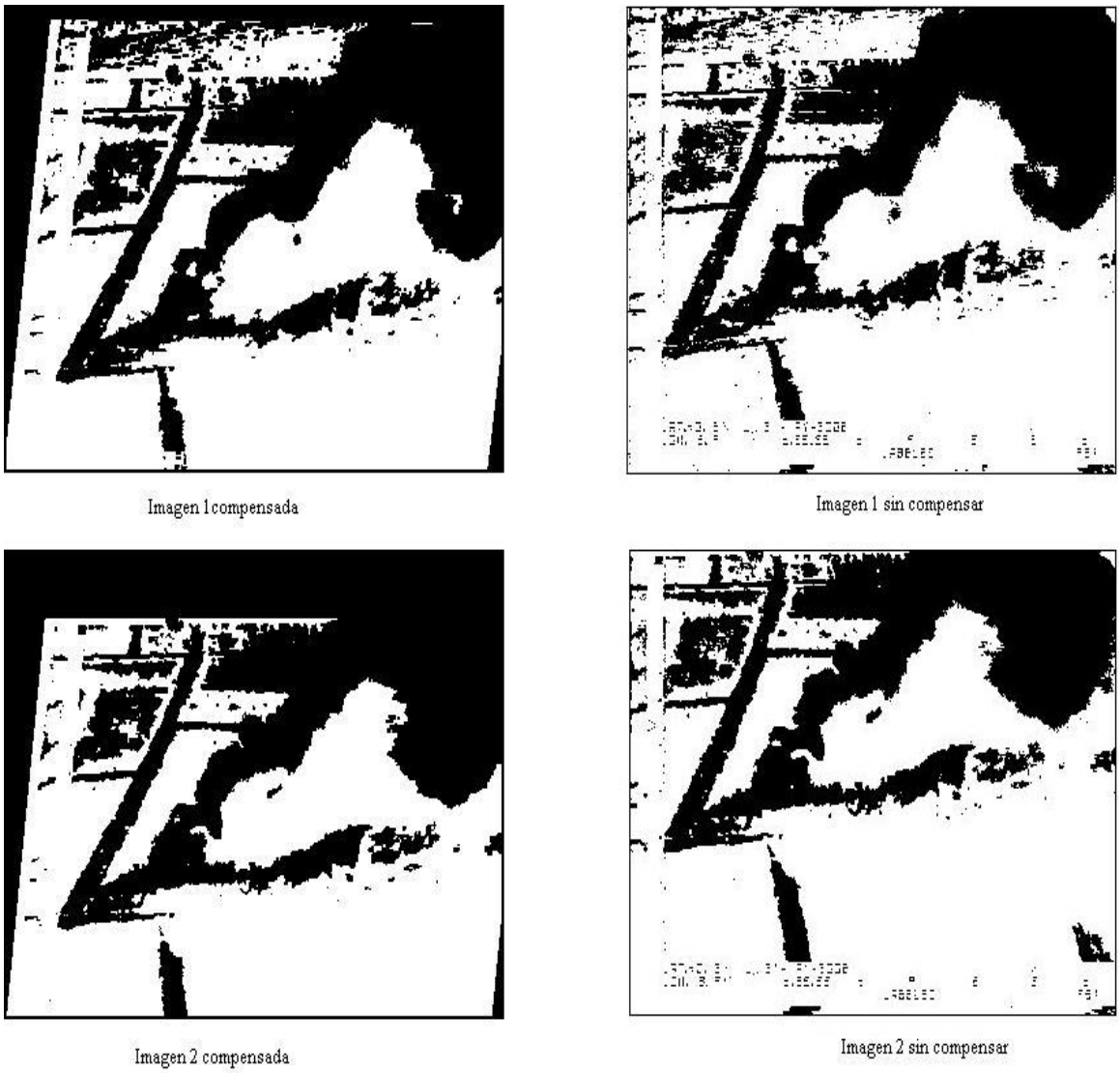


Figura 12.

3.3.3. Compensación de imágenes en blanco y negro.

La compensación de imágenes en blanco y negro se complica un poco más debido a que en este caso tenemos una gran variedad de valores para los píxeles. La nueva posición de los píxeles se obtiene también aplicando la transformación de la matriz de homografía sobre la posición de éstos.

En este caso, para poder resolver cual es el valor que se le asigna a los valores de los píxeles de la imagen transformada usaremos dos posibles métodos:

- *Aproximación bilineal*. Esta aproximación interpola entre todos los valores de los píxeles circundantes para poder obtener el valor de intensidad.

- *Aproximación del píxel más parecido*. Esta aproximación es matemáticamente más sencilla, ya que no es necesario realizar interpolaciones de ningún tipo sobre los datos, simplemente se toma, de entre los cuatro que rodean al píxel destino, aquel que más se parece.

  La compensación de imágenes en blanco y negro resultará más lento que la de imágenes binarias debido a que por cada píxel será necesario hacer un mayor número de operaciones. Con el fin de reducir en lo posible estos tiempos de cómputo, más adelante se verán diferentes métodos que nos permiten reducir el procesamiento.

3.3.3.1. Aproximación bilineal.

La forma de proceder en este caso consiste en lo siguiente. Partimos de la imagen compensada en la iteración anterior. Poseemos la matriz de homografía que nos traslada desde la imagen compensada anterior hasta la imagen actual.

Para cada píxel de la imagen compensada hallamos la posición transformada en la imagen actual. Dicha posición poseerá decimales, así que nos quedamos con la parte entera de dicha posición, guardando la parte decimal de la misma. Este proceso se puede ver en la Figura 13. En dicha figura podemos ver como el valor que se le asigna al píxel de la imagen compensada es el resultante de interpolar de forma bilineal usando como valores los de las cuatro intensidades que rodean al píxel transformado.

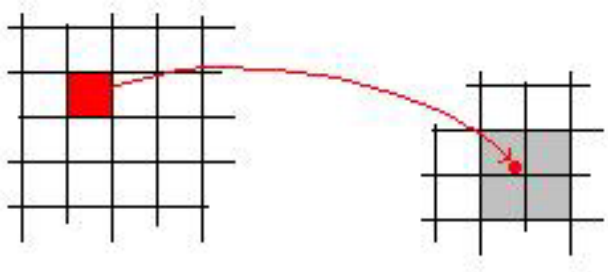


Figura 13.

Conocida las intensidades de los píxeles que rodean al píxel transformado, al píxel original se le asignará el valor dado por:

```
f_def = floor(fip)
c_def = floor(cip)
df = fip - f_def
dc = cip - c_de;
c1 = (1. - df)
c4 = c1*dc
c2 = (1. - dc)
c1 *= c2
c2 *= df
c3 = df*dc
valpix = c1*nw +c2*sw +c3*se +c4*ne
```

Siendo nw, ne, sw y se los valores de las intensidades de que rodean al píxel transformado.

Una vez establecido el valor de intensidad asociado al píxel se pasa al siguiente píxel, siguiendo un proceso igual al ya especificado.

Es de señalar que como resultado de aplicar la aproximación bilineal sobre la imagen para hacer la compensación, se producirá un ligero filtrado de la imagen, lo cual puede generar problemas posteriores si se pretende analizar digitalmente la imagen compensada resultante. En la Figura 14 podemos encontrar una imagen que a sido resultado de la compensación con aproximación bilineal, vemos como se ve un poco difusa debido al filtrado.

3.3.3.2. Aproximación del píxel más parecido.

Inicialmente, esta aproximación es igual a la anterior, es decir, se sigue el mismo método de transformar la posición del píxel de la imagen compensada previamente y estudiar los cuatro píxeles que circundan la posición transformada. La diferencia viene dada en la forma de elegir la intensidad del píxel en la imagen compensada.

Figura 14.

Conocidas las intensidades de los cuatro píxeles que rodena al píxel de la imagen compensada transformado debemos elegir la que más se parezca a la intensidad del píxel de la imagen compensada en la iteración anterior del algoritmo. Para elegir dicha intensidad, hallamos la distancia entre la intensidad del píxel de la imagen compensada

en la iteración anterior y las intensidades de los cuatro píxeles vecinos en la imagen actual. Tomaremos la intensidad que este a menor distancia de la intensidad del píxel de la imagen compensada en la iteración anterior.

De esta manera conseguimos reducir casi al mínimo los posibles errores numéricos producidos en el cálculo de la homografía, ya que el propio algoritmo de compensación de imágenes nos permite corregir errores en torno a un píxel.

Análisis comparativos entre este método y el anterior indican que, teniendo aproximadamente la misma carga computacional, el método de aproximación por el píxel más parecido genera mejores resultados, al margen de no producir efecto de filtrado sobre la imagen. En la Figura 15 podemos ver las mismas imágenes que en la Figura 14, pero compensadas mediante el método del píxel más cercano.

3.3.4. Compensación de imágenes en color.

La compensación sobre imágenes en color tiene el mismo proceder que sobre imágenes en blanco y negro, solo que en este caso los valores de intensidad no vendrán dados por un solo nivel de intensidad, sino por tres: Rojo, verde y azul.

Debido a que ahora cada píxel esta representado por una intensidad en cada plano de color, las distancias entre intensidades de píxel deberemos calcularlas mediante la distancia euclídea entre ambos píxeles.

Como veremos a continuación se produce un aumento notable del número de operaciones debido a que ahora debemos realizar el trabajo de la compensación en blanco y negro casi por triplicado.

Al igual que en la compensación en blanco y negro, en este caso tenemos también dos posibilidades a la hora de hacer la compensación de la imagen, mediante la aproximación bilineal o mediante la aproximación del píxel más parecido.

3.3.4.1. Aproximación bilineal.

La aproximación bilineal en este caso se hace de la misma forma que en el caso de banco y negro, solo que ahora aplicaremos la aproximación a cada uno de los planos por separado, dando lugar a un color que se puede aproximar por la interpolación bilineal del píxel en color.

Figura 15.

El problema que se nos presenta en este caso es que, a pesar de todo, la interpolación de los tres planos de color cada uno por separado en realidad no genera la verdadera intensidad RGB. Para poder hacerlo de forma correcta deberíamos tener en cuanta los tres planos de color al mismo tiempo, de modo que se tomara el color que

más se aproximara a la mezcla de los tres planos. Por desgracia esto nos supondría una gran carga computacional. Teniendo en cuanta que los resultados aplicando el método a cada plano por separado son bastante buenos, se ha decidido usar este método, al ser más sencillo y rápido. En la Figura 16 podemos encontrar un ejemplo de la forma de actuar del proceso.

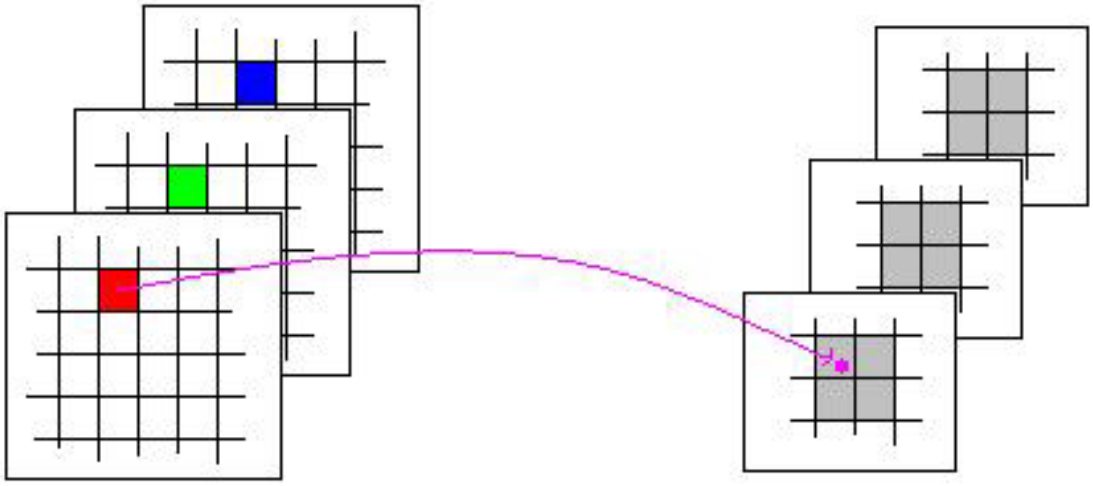


Figura 16.

Por último señalar que, como efecto de la aproximación bilineal se producirá un filtrado paso bajo sobre la imagen que generará un pequeño emborronamiento de la misma. Esta característica es intrínseca al método que estamos usando, de modo que es inevitable. En la Figura 17 podemos encontrar una secuencia de imágenes estabilizadas en color.

3.3.4.2. Aproximación del píxel más parecido.

En este caso, a pesar de ser conceptualmente igual que al píxel más parecido de blanco y negro, matemáticamente se vuelve un poco más complejo.

Figura 17.

En blanco y negro, dos píxeles son muy parecidos cuando sus intensidades son muy cercanas, poseen un valor numérico cercano, pero en RGB la cosa se complica mucho más. En este caso, podemos decir que dos píxeles son muy parecidos si la distancia euclídea en el espacio RGB de cada uno de ellos es pequeña, pero en realidad

esto no ocurre de esta forma, la percepción humana actúa de forma diferente. Veámoslo con un ejemplo, supongamos los tres píxeles en el espacio RGB de la Figura 18.

En dicha figura podemos ver como tenemos tres píxeles separados por una distancia D1-2 y D2-3. Si aplicamos la aproximación de que el píxel más parecido es el que está más cerca del píxel original, siendo el original el píxel P2, deberíamos tomar como píxel más parecido al píxel D3. El problema aparece cuando analizamos estos datos con la percepción real de una persona. Debido a que el espacio en RGB no es un espacio homogéneo, las distancias que obtenemos en dicho espacio no se corresponden con las distancias que notan los ojos humanos y se puede dar el caso de ser el píxel D1 en realidad el más parecido.

Para poder remediar esto es necesario pasar del espacio RGB a un espacio homogéneo que nos permita obtener las distancias de acuerdo con la percepción humana, que, a fin de cuentas, es la que verá las imágenes. Podemos encontrar diversos sistemas de referencia homogéneos y transformaciones de RGB a estos sistemas que nos permiten tener medidas fiables acerca de la distancia, así, para obtener una medida más exacta de las distancias sería necesario transformar a uno de estos espacios.

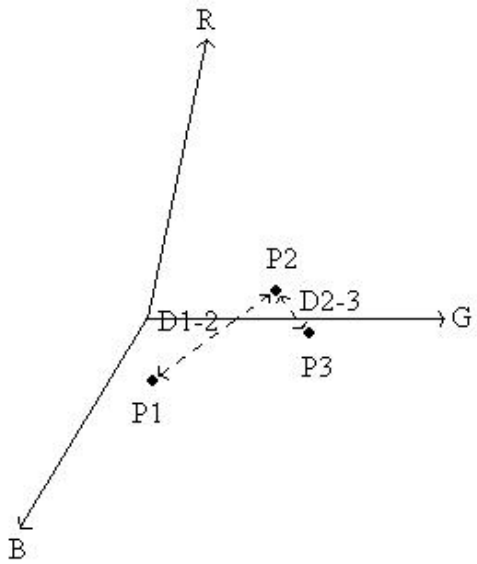


Figura 18.

El problema que se nos presenta transformando a uno de estos espacios es la importante carga computacional que esto supone, ya que es necesario realizar transformaciones sobre cada uno de los píxeles, además de las que ya se impone en la compensación de la imagen.

Por tanto, lo que haremos será tomar el píxel más cercano en el espacio RGB, a sabiendas que es incorrecto, ya que las diferencias no son muy apreciables y la carga computacional notablemente menor. Así, calcularemos la distancia entre el valor del

píxel de la imagen compensada en la iteración anterior con los valores de los cuatro vecinos, quedándonos con el píxel que posea una intensidad RGB más cercana. En la Figura 19 podemos encontrar una secuencia estabilizada de esta manera.

Figura 19.

3.3.5. Reducción de tiempo de procesamiento en la compensación de imágenes.

Se ha dejado entre ver en las secciones anteriores que uno de los principales problemas a los que nos enfrentamos al hacer la compensación de una imagen es el tiempo de procesamiento necesario para llevarla a cabo. Este es un factor determinante que puede ser de gran importancia, ya que en algunos casos este tiempo puede suponer poco menos de la mitad del tiempo necesario para compensar una imagen completa.

No esta demás que estudiemos posibles sistemas de compensación que nos permitan una reducción apreciable en el tiempo de procesamiento del mismo.

Queda claro que, sea como sea, lo único inalterable en el procedimiento de la compensación será la transformación realizada sobre la posición de los píxeles. Esto no lo podemos cambiar, ya que es la esencia misma de la compensación de la imagen, pero si podemos aplicar una serie de ideas que se desprenden del estudio de las imágenes compensadas. A continuación veremos algunos métodos implementados en el algoritmo de compensación y las razones de su uso.

3.3.5.1. Selección de la región sobre la cual se aplicará la compensación de la imagen.

Si se analizan las imágenes una vez que se la aplica la compensación, veremos que si el movimiento de una imagen a la anterior es muy grande aparecen grandes zonas negras en los bordes de la imagen, estas son zonas donde no existe imagen y por tanto se pintan de negro, ya que no hay datos acerca de ella. Lo podemos ver en la figura 20.

El problema que suponen estas zonas negras de los borde es que el algoritmo de compensación ha tenido que ir mirando cada uno de estos píxeles negros de los bordes y transformarlos para poder saber que dicho píxel al transformarlo se sale fuera del área de la imagen y, por tanto, no existe información, por lo que se pone a color negro. En la Figura 21 podemos ver con más claridad el porque de los borde negros de la imagen.

Una forma de remediar esto es sabiendo desde el principio cuales van a ser los píxeles que al ser transformados van a caer dentro del área de la imagen. En el calculo de cual es esta zona se invertirá un tiempo, pero con total seguridad este tiempo será inferior al del cálculo de los píxeles del borde negro.

Figura 20.

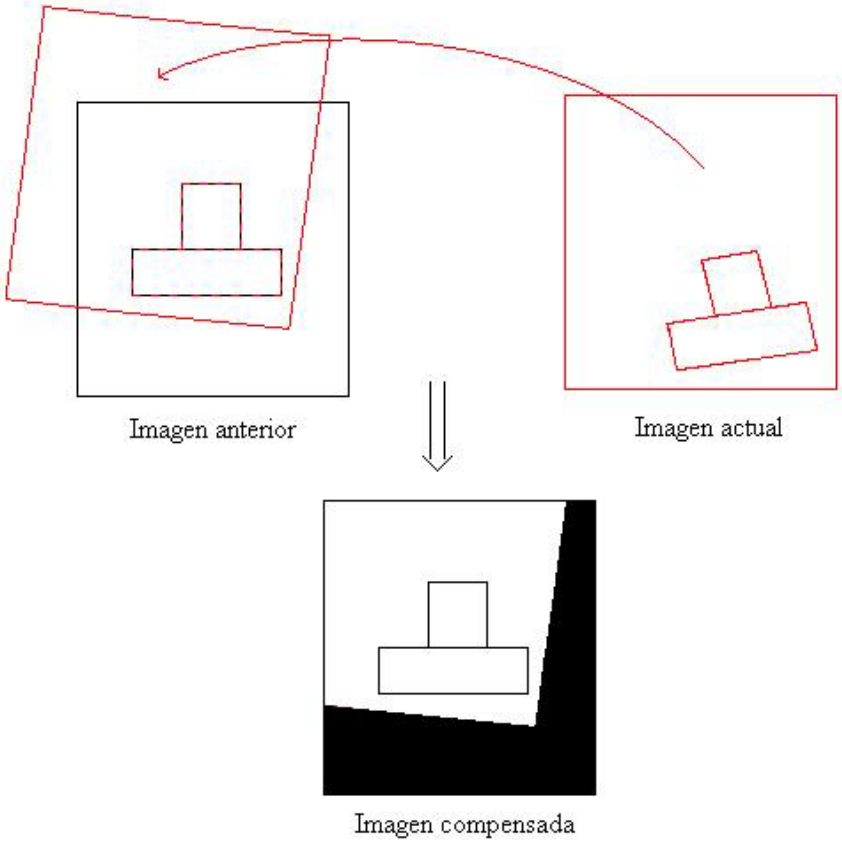


Figura 21.

En la Figura 21 podemos ver que cuando trasladamos la imagen actual sobre la anterior se definen unos márgenes a partir de los cuales la imagen compensada se pone de color negro debido a la falta de información. Por tanto podríamos de alguna forma hallar estos márgenes y realizar solo la compensación de la zona de la imagen que se encuentra dentro de ellos.

Para conseguir esto no tenemos más que obtener la transformación inversa de la matriz de homografía y aplicársela a los cuatro bordes de la imagen de la imagen actual, para saber donde van a ser colocados en la imagen compensada.

Si aplicamos este método al ejemplo de la figura 21 veremos que la única zona a la cual debemos aplicar la compensación es la que se muestra en la Figura 22, marcada de color gris. En consecuencia nos estamos ahorrando el procesamiento de hacer la compensación a toda la zona marcada con negro.

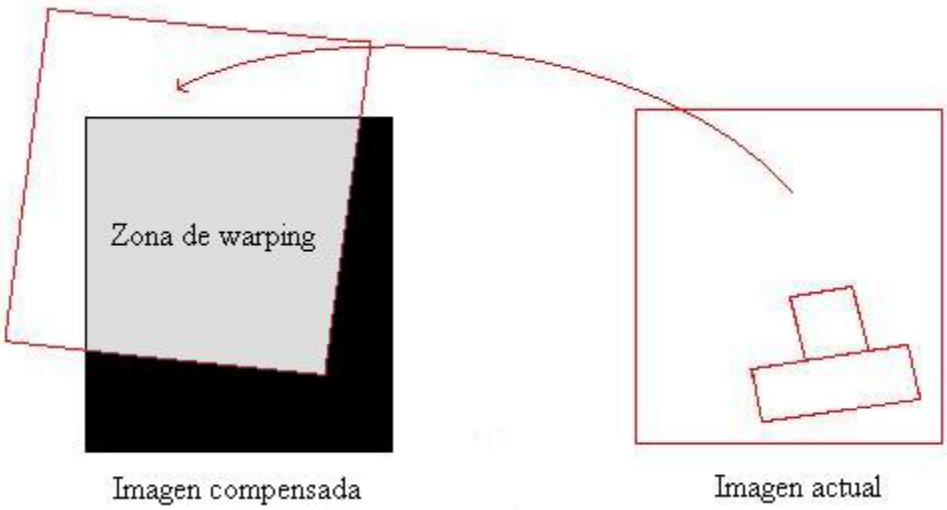


Figura 22.

Resumiendo podemos reducir el procedimiento a lo siguiente:

- Obtener la inversa de la matriz de homografía H^{-1}.
- Aplicar H^{-1} a la posición de los píxeles de las cuatro esquinas de la imagen actual.
- Unir con rectas los cuatro puntos transformados definiendo la zona a compensar.
- Aplicar la compensación solo a la zona definida anteriormente.

Vemos como este procedimiento no requiere a penas capacidad computacional, ya que la matriz de homografía, al ser de tamaño definido y fijo (3x3), se puede invertir directamente aplicando el método de la traspuesta del adjunto de la matriz.

El uso de este método es muy aconsejable cuando el movimiento de una imagen a la siguiente es muy grande, ya que se producirán grandes zonas negras en lo bordes y, por tanto, la zona a compensar se reducirá notablemente. Por otro lado no es muy aconsejable su uso cuando los movimientos en la imagen son superfluos o pequeños, ya que en ese caso la zona de compensación es prácticamente toda la imagen y no merece la pena invertir procesamiento en hallarla.

3.3.5.2. Compensación selectiva.

Como veremos más adelante, la estructura de clases que se ha utilizado para implementar los algoritmos encapsula a la imagen en objetos derivados de la clase "CImage", la cual posee un atributo que se denomina "roi" (Region Of Interes). Así, a cada imagen le podemos definir una zona en el cual estamos especialmente interesados.

Una forma de aprovechar esta "roi" es haciendo la compensación solo de la zona indicada y no de toda la imagen. De esta manera nos ahorraremos procesamiento, ya que, si el usuario no necesita el resto de la imagen, es un desperdicio de procesamiento el realizar la compensación de esta.

Esta es una forma sencilla que reducir procesamiento que no supone ninguna perdida de generalidad por parte del algoritmo. Un ejemplo lo podemos ver en la Figura 23.

Como resulta lógico, no es posible cambiar la "roi" en pleno funcionamiento del algoritmo de compensación ya que, al cambiar la dimensión de las imágenes la compensación dejaría de funcionar correctamente debido a cambios en el sistema de referencia de las imágenes.

Secuencia original

Secuencia de una región compesada

Figura 23.

3.3.5.3. Compensación con interpolación por rectas.

En término general podemos decir que para transformar la posición de un píxel es necesario realizar las siguientes operaciones:

$$H = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix}$$

$$\begin{aligned} k &= x * H[6] + y * H[7] + H[8] \\ x' &= (x * H[0] + y * H[1] + H[2])/k \\ y' &= (x * H[3] + y * H[4] + H[5])/k \end{aligned}$$

Luego las operaciones matemáticas necesarias para poder transformar un píxel serán:

- 6 multiplicaciones.
- 6 sumas.
- 2 divisiones.

Vemos como la carga computacional es bastante elevada, sobre todo por el número de divisiones, ya que los microprocesadores comerciales no están preparados para hacer esta operación en un solo ciclo en hardware, necesitando un número de ciclos considerable para poder implantar el algoritmo que resuelva la operación.

Como consecuencia de esto nos sería útil poder simplificar el problema de la transformación de los píxeles mediante la matriz de homografía. A la hora de buscar posibles métodos matemáticos podemos diferencias dos casos:

- *Matriz de homografía afín*. En este caso, si tenemos en cuenta que la matriz posee la siguiente forma:

$$H = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix}$$

Podremos simplificar las ecuaciones, llegando a lo siguiente:

$$x' = x * H[0] + y * H[1] + H[2]$$

$$y' = x * H[3] + y * H[4] + H[5]$$

Vemos como hemos disminuido notablemente el número de operaciones, ya que en este caso son:

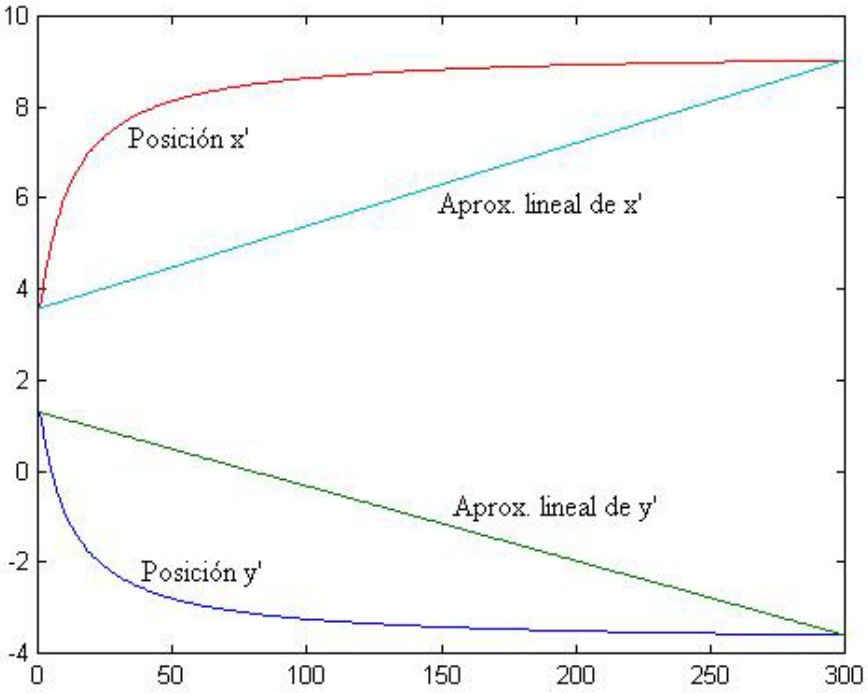
- 4 multiplicaciones.
- 4 sumas.
- 0 divisiones.

Hemos logrado eliminar las divisiones, las cuales eran el mayor lastre en el procesamiento y además hemos conseguido disminuir el número de multiplicaciones y sumas en 2 cada uno.

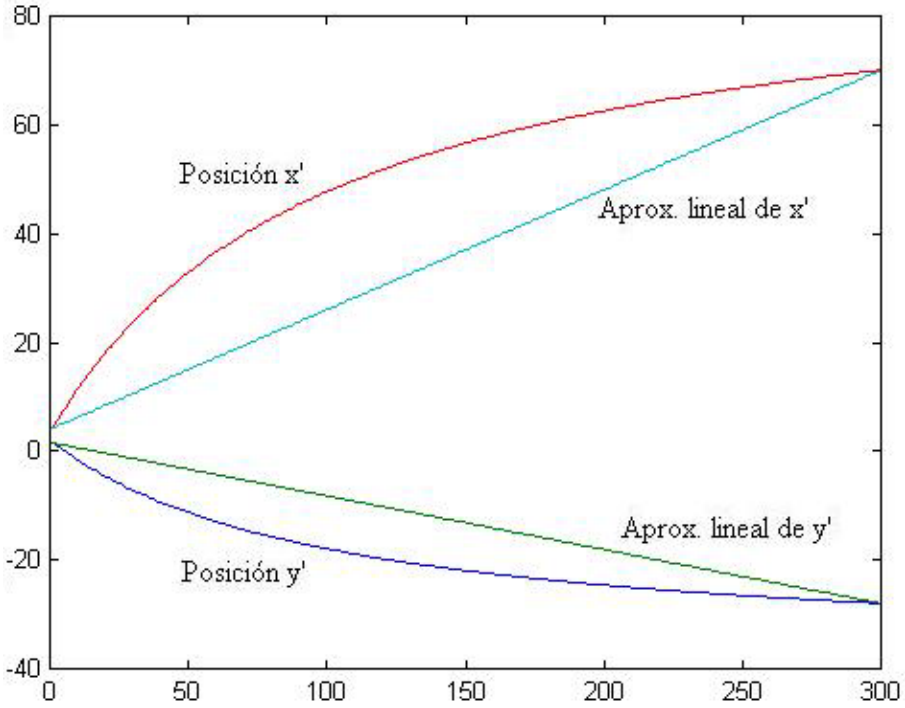
- *Matriz de homografía no afín*. En este caso no podemos hacer ningún tipo de simplificación sobre las ecuaciones, de modo que el procesamiento es el mismo.

El problema que se nos plantea es que la mayoría de las transformaciones no son afines, de modo que la reducción de tiempo se puede hacer en pocas ocasiones. A pesar de todo, en función de las circunstancias, se podría hacer una aproximación. Si los valores de H[6] y H[7] son cercanos a cero podemos suponer que la matriz de homografía es afín y por tanto aplicar las simplificaciones anteriores. El problema es saber cual es la cota a partir de la cual se puede considerar que estos dos elementos de la matriz son nulos. Para poder saberlo se han hecho una serie de experimentos analizando el distanciamiento entre la H aproximada y la H original. Así:

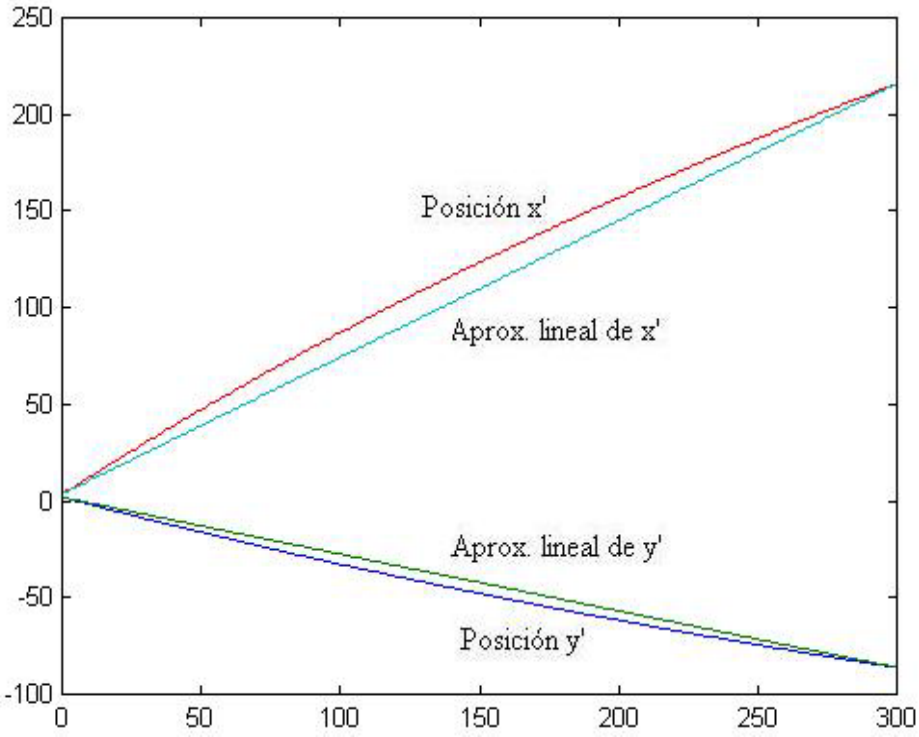
$$H = \begin{bmatrix} 0.9249 & -0.3801 & 1 \\ 0.3801 & 0.9249 & 3 \\ 0.1 & 0.1 & 1 \end{bmatrix}$$



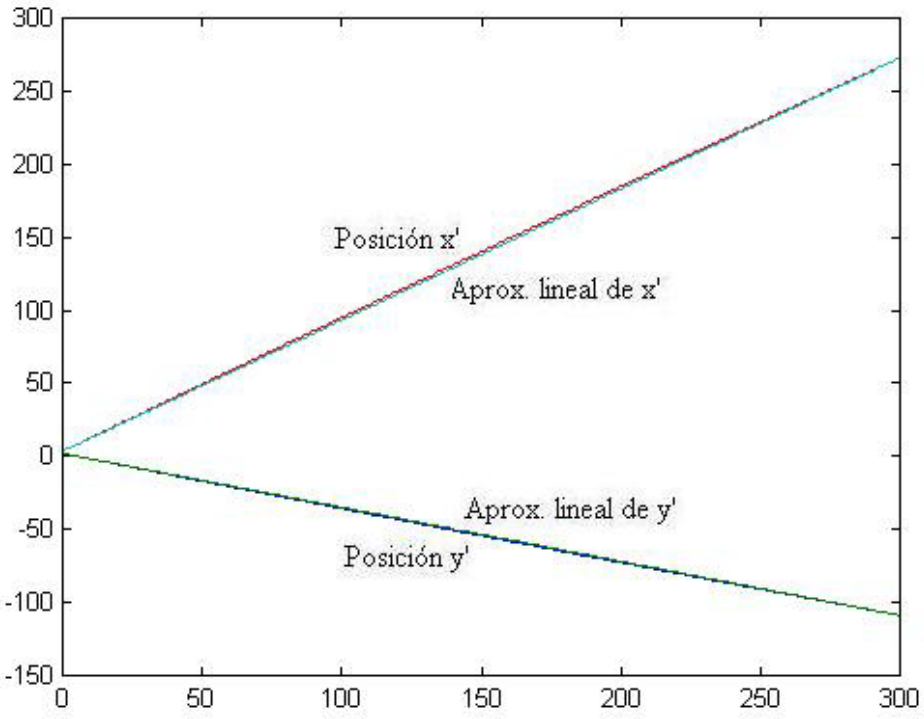
$$H = \begin{bmatrix} 0.9249 & -0.3801 & 1 \\ 0.3801 & 0.9249 & 3 \\ 0.01 & 0.01 & 1 \end{bmatrix}$$



$$H = \begin{bmatrix} 0.9249 & -0.3801 & 1 \\ 0.3801 & 0.9249 & 3 \\ 0.001 & 0.001 & 1 \end{bmatrix}$$



$$H = \begin{bmatrix} 0.9249 & -0.3801 & 1 \\ 0.3801 & 0.9249 & 3 \\ 0.0001 & 0.0001 & 1 \end{bmatrix}$$



De los experimentos podemos extraer que, si la matriz de homografía esta normalizada respecto de H[8], es decir, si H[8]=1, y H[6] y H[7] son menores que 10^{-4}, la matriz de homografía se puede aproximar por una matriz afín, pudiéndose aplicar las reducciones anteriormente especificadas par matrices afines en el cálculo de la posición transformada del píxel.

Vemos como hemos conseguido extrapolar las simplificaciones de las matrices afines un poco más allá de estas, pero esto no es suficiente, ya que, aun haciendo la suposición anterior, las mayoría de las matrices de homografía no la cumplen, por lo que sería interesante usar un método que nos permitiera reducir el tiempo de computo que no sea dependiente del tipo de matriz de homografía. Una posibilidad podría ser aproximar la transformación de forma lineal. La aproximación lineal consiste en calcular la transformación del primer y último píxel de cada fila, obteniéndose la recta que va desde la posición inicial a la final de x' e y'. Calculada dicha recta, la posición x' e y' del resto de píxeles intermedios se calculará sustituyendo en las ecuaciones halladas para cada fila. Así las operaciones necesarias para cada píxel serán:

$$x' = ax + b$$
$$y' = cy + d$$

Si suponemos el ancho de la imagen lo suficientemente elevado podremos despreciar los cálculos para la obtención de las rectas por cada fila, de modo que el número aproximado de operaciones matemáticas por píxel sería el siguiente:

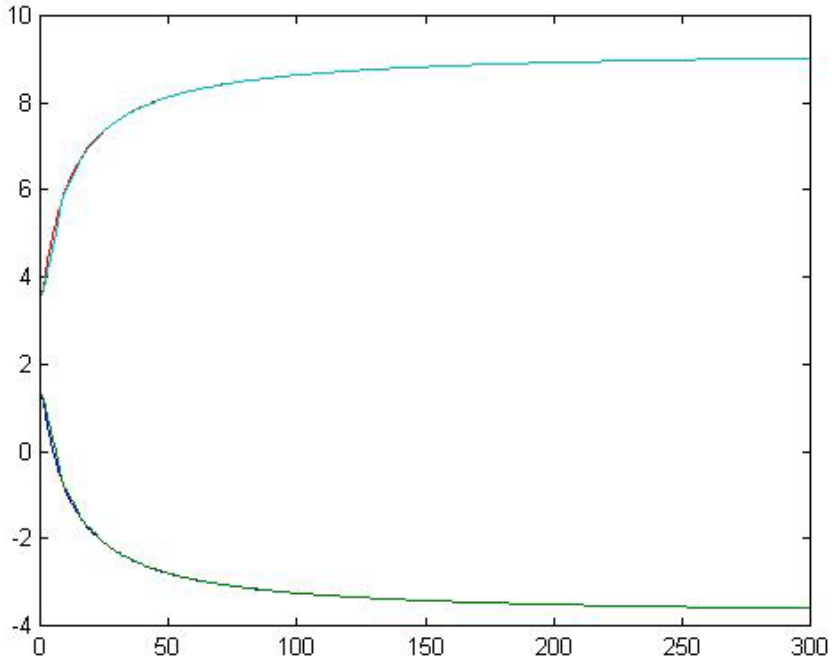
- 2 multiplicaciones.
- 2 sumas.
- 0 divisiones.

Vemos como el número de operaciones por píxel es inferior a los necesarios usando una transformación afín, solo que en este caso, este método es independiente del tipo de transformación, afín o no.

Debido a que la transformación no tiene porque ser lineal, podemos hacer genérico el método haciendo una linealización por tramos de la transformación, ajustándose a la transformación lo suficiente como para no generar errores demasiado apreciables. Así, en función de la no linealidad de la transformación se crearan mayor o menor cantidad de tramos linealizados. Veamos ejemplos de cómo la linealización se ajusta a la transformación:

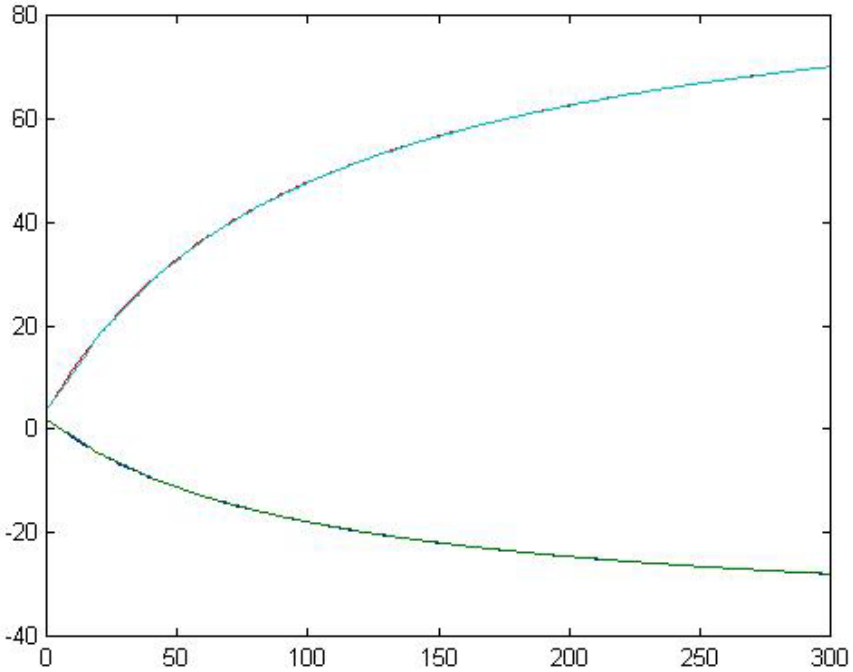
$$H = \begin{bmatrix} 0.9249 & -0.3801 & 1 \\ 0.3801 & 0.9249 & 3 \\ 0.1 & 0.1 & 1 \end{bmatrix}$$

Tramos = 40

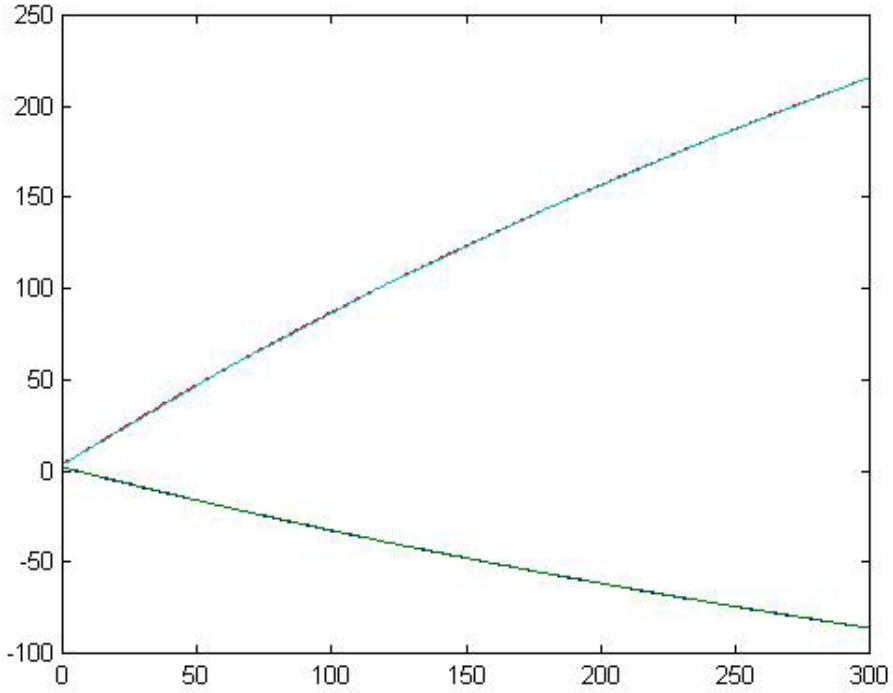


$$H = \begin{bmatrix} 0.9249 & -0.3801 & 1 \\ 0.3801 & 0.9249 & 3 \\ 0.01 & 0.01 & 1 \end{bmatrix}$$

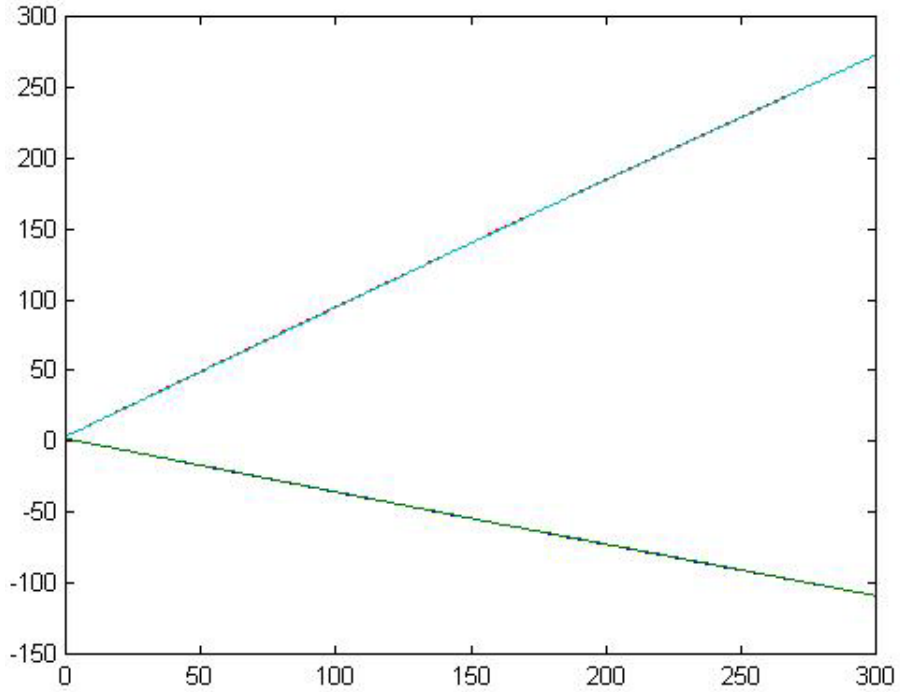
Tramos = 15



$$H = \begin{bmatrix} 0.9249 & -0.3801 & 1 \\ 0.3801 & 0.9249 & 3 \\ 0.001 & 0.001 & 1 \end{bmatrix}$$ Tramos = 5



$$H = \begin{bmatrix} 0.9249 & -0.3801 & 1 \\ 0.3801 & 0.9249 & 3 \\ 0.0001 & 0.0001 & 1 \end{bmatrix}$$ Tramos = 2



En todos los ejemplos están calculados el número de tramos de linealización de modo que el error máximo sea aproximadamente 1 píxel, el cual es un buen error máximo. Vemos como el número de tramos necesario para poder disminuir el error aumenta de forma casi cuadrática

Resulta interesante poder disponer de una función que se autoajuste a las circunstancias de la transformación. Esto resultaría sencillo, ya que no hay más que obtener la parábola de regresión a partir de los valores que se han obtenido en los ejemplos anteriores. Con esto conseguiremos una ecuación cuadrática en función del valor máximo entre H[6] y H[7] que nos devolverá el número adecuado de tramos de linealización.

Debemos tener en cuanta que el número de tramos no debe ser demasiado elevado, ya que en caso contrario la sobrecarga generada por las opresiones para hallar las rectas deja de ser despreciable. Una cota adecuada sería hacer igual el número máximo de tramos al ancho de la imagen dividido entre 80. Si el número de tramos supera esa cota, entonces calcularíamos la transformación directamente, usando la matriz de homografía sin linealizar.

Capítulo 4

Implementación y estructuración en clases

4.1. Introducción.

La codificación de los algoritmos se ha hecho en ANSI C. Las razones por las que hemos usado este lenguaje y no C++ son básicamente las siguientes:

- Lenguaje de programación completamente estándar.
- Lenguaje de más bajo nivel que C++, lo cual genera que el código sea más óptimo.

Como principal inconveniente encontramos que ANSI C es un lenguaje poco estructurado en comparación con C++. El problema de la estructuración la resuelve C++ mediante el uso de clases.

Con la intención de tener lo mejor de los dos lenguaje de programación y aprovechando que C++ soporta ANSI C, se han codificado los algoritmos en ANSI C, estructurándolos lo más posible y evitando las variables globales y demás prácticas de la programación que dificultan el procesamiento multihilo y la estructuración. Una vez hechas las funciones en ANSI C se ha creado una interfaz en C++, fácil de manejar y con todas las comodidades de la programación orientada a objeto.

De esta manera hemos conseguido un código optimizado en C y encapsulado en una interfaz C++ que facilita su uso en otros programas sin más que crear un objeto de la clase adecuada.

Todos los objetos pretenden ser autocontenidos, de modo que sea suficiente con incluir la cabecera en la que está definida la clase.

En los puntos que siguen vamos a hacer una descripción detallada de cada una de las clases creadas para dar soporte a la compensación de imágenes.

Buena parte de la documentación en la que nos hemos basado para poder generar clases de la forma más estándar posible y de acuerdo con las características de C++ la encontramos en [3]

4.2. Clase CTracking

Esta clase encapsula el conjunto de funciones necesarias para poder hacer el seguimiento de ventanas. Antes de comenzar a describir la clase es necesario

especificar levemente como funcionan el conjunto de funciones de seguimiento para poder comprender la estructura de la clase.

Después de describir levemente las funciones que nos permiten realizar el seguimiento de ventanas se pasará a la descripción de los métodos y atributos de la clase que estamos estudiando.

4.2.1. Funciones ANSI C de la clase CTracking.

Para comenzar, debemos establecer cual debe ser el orden de llamadas de las funciones disponibles para poder realizar seguimiento sobre un conjunto de imágenes. Los pasos a seguir son los siguientes:

- Como primer paso será necesario hacer las reservas de memoria adecuadas, tales como el buffer de la imagen. Además debemos definir una serie de estructuras en las cuales están contenidos los parámetros del seguimiento de ventanas.

- Una vez que todos los parámetros del seguimiento están inicializados debemos realizar el primer paso de seguimiento sobre la primera imagen que nos llegue. El primer paso de seguimiento es diferente del resto ya que es necesario definir por primera vez las ventanas que se van a seguir en las imágenes siguientes, en el primer paso de seguimiento todas las ventanas son nuevas. Para este primer paso usaremos la función mk_lsec() para poder crear la lista de secuencia de ventanas y la función trk_iteracion() con su tercer parámetro a 1, indicando que es el primer paso de seguimiento.

- Una vez que se ha inicializado el seguimiento, en cada iteración debemos usar la función trk_iteracion() con su tercer parámetro a 0.

- Cuando el número de iteraciones sea igual al tamaño de la lista de secuencia deberemos usar la función shift_lsec() para desplazar una posición a la izquierda la lista de secuencia de ventanas.

Conocido el orden de llamada a las funciones vemos una leve descripción de las funciones anteriormente especificadas y de los parámetros necesarios:

- c_param

Esta es una tipo de estructura donde están guardados todos los parámetros necesarios para el funcionamiento del seguimiento. Veamos los elementos de esta estructura:

int fil_in;	Fila inicial del área de interés
int col_in;	Columna inicial del área de interés
int anchoi_i;	Ancho del área de interés
int altoi_i;	Alto del área de interés
int anchoi;	Ancho de la imagen completa
int altoi;	Alto de la imagen completa
int anchov;	Ancho de ventana
int altov;	Alto de ventana
int traza;	Nivel del trazador: Anulado, medio, alto
int display;	Activación del display
int filtrar;	Activación del filtro
int anchof;	Ancho de la ventana de filtrado
int altof;	Alto de la ventana de filtrado
float sigma;	Sigma del filtrado gaussiano
int ver_sel;	Versión del algoritmo del selección de ventanas
int nivel_av;	Nivel del autovalor
int solape;	Máximo solape permitido entre ventanas
int nv_max;	Número máximo de ventanas
int si_presel;	Activa preselección
int ver_trk;	Versión del algoritmo de seguimiento
int corr_norm;	Activa correlación normalizada
int ver_corr;	Versión de correlación
float error_in;	Error máximo permitido al comienzo
float error_fin;	Error máximo permitido al finalizar
int radio;	Radio de búsqueda de la ventana
int niter_max;	Número máximo de iteraciones
float d_conv;	Mínimo desplazamiento para concluir
int tam_cls;	Tamaño mínimo del cluster
float radio_cls;	Radio máximo del cluster
float precic_cls;	Precisión mínima del ajuste
int radio_rep;	Radio de repesca de ventanas
int timout;	Timeout del seguimiento (0 = tiempo infinito)
int timout_it;	Timeout en iteraciones (0 = infinitas iteracioens)
int max_predic;	Máximo numero de ventanas de predicción en %
par_int par_cam;	Parámetros de la cámara que se está usando

Como ya se comentó previamente, algunos de estos parámetros son cruciales para obtener un buen seguimiento de ventanas, tales como: nv_max, anchov, altov, nivel_av, error_in, error_fin y algunos otros.

La inicialización de los parámetros dependerá de forma directa del tipo de imágenes que estemos tratando. En términos generales podemos hacer una distinción clara entre imágenes infrarrojas y visuales. En función de ser una de estos dos tipos podremos ajustar los parámetros de una forma u otra:

- *Infrarrojo.* Este tipo de imágenes suelen ser más borrosas que las visuales. Además, nos encontramos que las zonas calientes son claras candidatas a ser objetos en movimiento, por lo que no nos interesa asignar ventanas a estas zonas, ya que en caso contrario no seremos capaces de discernir entre el movimiento de la imagen al completo y el movimiento de objetos aislados en la misma.

  Para evitar esto, es conveniente que el nivel del autovalor no sea demasiado elevado, de modo que no se centren todas las ventanas en las zonas más calientes, que son las más luminosas y las que generan más contraste con el fondo. Al mismo tiempo, teniendo en cuenta que estas imágenes no suelen estar demasiado definidas, será necesario aplicar una mayor tolerancia en la búsqueda de ventanas.

  Con la intención de combatir la falta de definición de este tipo de imágenes podríamos incluso aumentar el tamaño de la ventana de seguimiento, a 9x9 por ejemplo, de modo que sea más fácil seguirlas.

  Podemos ver como en este tipo de imágenes es necesario un aumento del procesamiento del seguimiento por encima de lo normal. Esto esta plenamente justificado dada la calidad de las imágenes con la que se prevé trabajar.

  Es necesario señalar que el algoritmo solo es efectivo cuando la secuencia de imágenes infrarroja esta puesta en un rango del infrarrojo tal que permite captar información de toda la escena a la que se está enfocando, ya que en caso contrario serán muy complejo obtener información fidedigna acerca del movimiento de la imagen.

  En la Figura 24 podemos encontrar un claro ejemplo de cómo debe ser una imagen infrarroja (Imagen 1) y como no debe ser (Imagen 2) para que el proceso de seguimiento resulte efectivo.

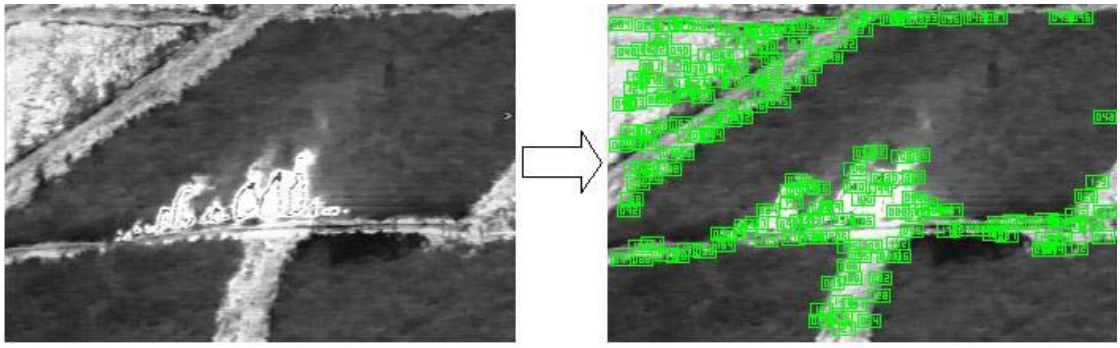


Imagen 1

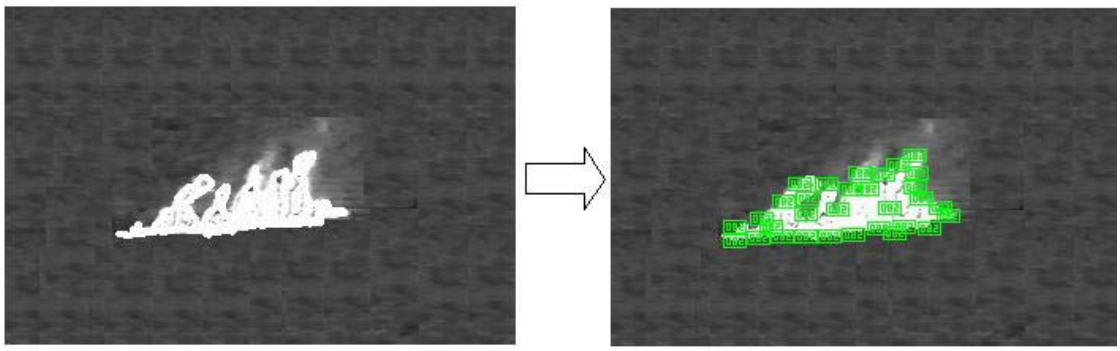


Imagen 2

Figura 24.

- *Visual.* En el caso en que las imágenes sean visuales, las posibilidades de hacer un buen seguimiento de ventanas son mayores. En este caso no corremos el riesgo de quedarnos en el plano de la imagen solo con una parte de la misma, independientemente del nivel de calor de los objetos de la escena, la cámara visual recibirá lo mismo.

  En términos generales, las imágenes visuales suelen tener más definición que las infrarrojas y suelen ser menos borrosas, por lo que no será necesario aumentar la tolerancia del seguimiento.

  Por otro lado, puede que se haga indispensable un filtrado sobre la imagen para evitar la aparición de ruido en la misma y para que los pequeños movimientos, los generados por el viento sobre las plantas por ejemplo, sean eliminados, mejorando el resultado del seguimiento de ventanas.

  El nivel del autovalor no conviene tenerlo demasiado elevado, debido a que en este tipo de imágenes el contraste existe, pero no suele ser demasiado acusado. Se recomiendo un valor en torno a 13.

Para imágenes visuales de muy poco contraste es necesario aumentar el tamaño de la ventana con el fin de facilitar la búsqueda de las ventanas en la imagen siguiente.

- par_sec_img
  Esta estructura contiene la información acerca de la secuencia de imágenes que se le pasa como parámetros. Esta estructura solo se usa cuando las imágenes a estabilizar proceden de un archivo de arrastre.

  Estos archivos de arrastres están compuestos solo por mapas de píxeles en escala de grises, de modo que cada octeto representa un píxel. Estos mapas de píxeles están colocados uno detrás de otro sin ningún tipo de cabecera para ahorrar tiempo de lectura de archivo y aumentar la eficiencia de la lectura, ya que todos poseen las mismas características.

  La estructura contiene los datos suficientes acerca del archivo de arrastre como para poder leerlo. Los datos que contiene son los siguientes:

int inic;	Imagen en la que comienza
int fin;	Imagen en la que acaba
int ancho;	Ancho de la imagen
int alto;	Alto de la imagen
int longitud;	Número de imágenes en el archivo de arrastre
char *nom_arch;	Nombre del archivo
int buff_lec;	Activa buffer de lectura
int buff_esc;	Activa buffer de escritura

- lis_sec
  La lista de secuencias agrupa todos los resultados del seguimiento hasta el instante considerado. El funcionamiento esperado es que se mantenga el orden temporal en p_hist, y que se desplacen las listas de posiciones a la izquierda cuando falte espacio.

  Esta es una de las estructuras más importantes del proceso de seguimiento, ya que guarda la información de seguimiento de iteraciones pasadas, hasta un número máximo definido por el usuario, hasta el momento presente.

  Esta estructura contiene una lista en la que poco a poco se va guardando la información acerca de la posición de las ventanas en pasos anteriores, calidad de la búsqueda, errores producidos, error en la búsqueda, ...

Gracias a que guarda gran cantidad de información, el algoritmo de seguimiento puede hacer búsquedas predictivas de las ventanas, analizando el movimiento en pasos anteriores de modo que se le pueda asignar una trayectoria y así averiguar cual puede ser el siguiente movimiento.

Los elementos de esta estructura son los siguientes:

int n_sec;	Numero máximo de secuencias
int step;	Ultimo paso realizado en p_hist
int *asg_nodo;	Lista de asignaciones a procs.
secuencia *p_sing;	Puntero a la sec. más singular
secuencia *p_sec;	Datos act. de cada secuencia
lis2_vent *p_hist;	Historia de posiciones
movi **ppos;	Historia de movimientos
lis_clus *p_lclus;	Lista de clusters
traza_trk **p_traza;	Datos -opcionales- de traza

Las funciones de seguimiento van rellenando la estructura en cada iteración, de modo que cuando esta está completa es necesario desplazarla a la izquierda para dejar espacio a nuevos datos.

- ippbuf
  Estructura que contiene la información suficiente acerca de una imagen: el mapa de píxeles, el ancho y el alto:

pixel ** pb;	Puntero al buffer de la imagen
int ancho;	Ancho de la imagen
int alto;	Alto de la imagen

Como peculiaridad encontramos que el buffer está apuntado por un puntero a puntero llamado “pb”. Las razones de este puntero a puntero vienen dadas por la necesidad de poder acceder a las filas de la imagen directamente sin necesidad de direccional usando el número de filas y el número de columnas. De esta manera “pb[0]” nos da un puntero a la fila 0 de la imagen y “pb[n]” nos da un puntero a la fila n de la imagen, facilitándonos la tarea de acceso a la imagen. En la Figura 25 podemos encontrar un gráfico explicativo.

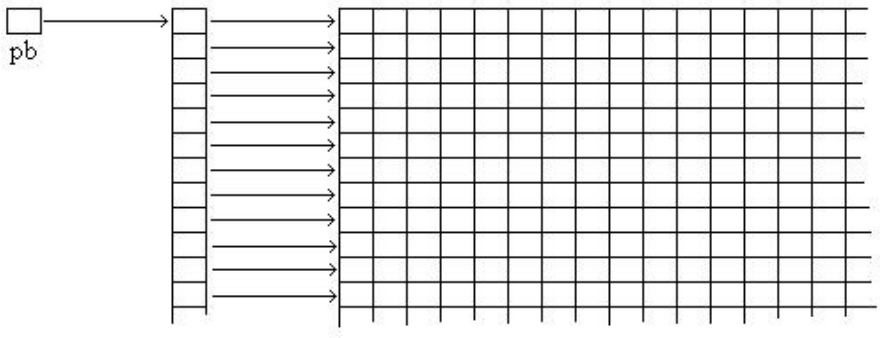


Figura 25.

- ippbuf *mk_ippbuf(int anchoi, int altoi)
  Esta función nos crea una estructura del tipo ippbuf, devolviéndonos un puntero a la misma.

  Como parámetros se le pasa el ancho y el alto de la imagen para que la función pueda hacer la reserva adecuada de memoria para el buffer.

- void del_ippbuf(ippbuf *p)
  Esta función libera la memoria reservada previamente mediante la función mk_ippbuf.

- lis_sec * mk_lsec(c_param *ppars, long nframe, lis_vent *plis)
  La función ml_lsec nos permite crear una lista de secuencia a partir de los parámetros que le pasamos como argumentos a la función.

  Devuelve un puntero a la lis_sec reservada dinámicamente.

  Como argumentos recibe:

  - Un puntero a la estructura de parámetros de configuración del seguimiento.

  - Entero indicando la longitud de la lista de secuencias. Su valor debe ser mayor de cero.

  - Puntero a una lista de ventanas creada con anterioridad por el usuario. Si no existe lista de ventanas se le pasa NULL indicando que las ventanas iniciales las tiene que crear la función.

- void del_lis_sec(lis_sec *p)
  Función que libera la memoria reservada para la lista de secuencia que se le pasa como parámetro.

- void trk_iteracion(lis_sec *lsec, c_param *ppars, ippbuf *img_nueva, int auto_ini)
  Función que realiza un paso de seguimiento sobre la imagen que se le pasa como parámetro. Esta función necesita que la lista de secuencias que se le pasa esté ya creada.

  Los parámetros que recibe son los siguientes:

  - Lista de secuencias con los datos de los pasos anteriores de seguimiento en el caso en que los hubiera.

  - Parámetros de configuración del seguimiento.

  - Estructura ippbuf con los datos de la imagen.

  - Entero cuyos únicos valores pueden ser 0 o 1. Si es 1 se le indica a la función que este el primer paso de seguimiento y que debe seleccionar las nuevas ventanas. Si su valor es 0 se le indica a la función que este no es el primer paso de seguimiento y que debe proceder con normalidad, buscando las ventanas en la nueva imagen y reponiendo las que se han perdido en proceso de búsqueda.

- lis_sec *serie_cbk2(lis_sec *psec, int nsteps, c_param *ppars, pixel **p, lis_vent *p_inlis, int niter)
  Esta función realiza lo mismo que la función trk_iteacion(...), solo que a esta función se le pasa directamente el número de iteraciones y ella se encarga de hacerlas todas de forma automática, desplazando las lista de secuencia de forma automática y generando todos los datos necesarios.

- void shift_lsec(lis_sec *psec, int n)
  Desplaza la lista de secuencia que se le pasa como parámetro una cantidad de posiciones hacia la izquierda dada por el parámetro “n”.

4.2.2. Métodos y atributos de la clase CTracking.

En la clase CTracking podemos encontrar los siguientes atributos:

- CList<REGION,DOUBLE,LINEAR> *RegionList

Este atributo es una lista de regiones que contiene información acerca de la posición de las ventanas y de si vale o no para poder obtener la matriz de homografía a partir de ésta.

Cada elemento de la lista es una REGION, la cual es una clase en la cual se guardan los siguientes datos:

```
POINT2D pos;          Position of the region on the image plane.
POINT2D prevPos;      Previous position of the region on the image plane.
POINT2D initPos;      Initial position of the region on the image plane.
int regionSize;       Tamaño de la región
unsigned char *pixels; Region en píxeles
double error;         SSD error
bool use_homography; Para ver si se usa al calcular la compensación.
int seqNumber;        Número de secuencia
```

De modo que por cada ventana que se haya seguido en el seguimiento generaremos uno de estos elementos en la lista.

- int LongHistory
  Especifica la longitud de lista de secuencias. Este es un valor constante a lo largo del procesamiento de seguimiento de una secuencia de imágenes. Si se quiere cambiar es necesario resetear el seguimiento.

- c_param *ptrk
  Este atributo guarda la configuración del seguimiento a lo largo del procesamiento.
  La clase está estructurada para poder crear funciones que permitan la modificación de algunos de los parámetros de seguimiento que son modificables en plena ejecución. Así, para poder modificarlos en pleno procesamiento será necesario crear una clase hija que herede públicamente a partir de CTracking, en esta clase hija se podrá crear una función que pueda acceder a los datos modificables de la clase.

  Los campos de significativos c_param que se pueden modificar en plena ejecución sin riesgo para el procesamiento de seguimiento son los siguientes:

```
int  traza;           Nivel del trazador: Anulado, medio, alto
int nivel_av;         Nivel del autovalor
float error_in;       Error máximo permitido al comienzo
float error_fin;      Error máximo permitido al finalizar
```

- par_sec_img *psec
  La clase puede ser configurada para recibir imágenes desde archivos de arrastre o bien para pasarle las imágenes una a una.

  Si las imágenes se le pasan desde un archivo de arrastre es necesario aportar los parámetros del mismo.

- lis_sec *plsec
  Puntero a la lista de secuencias. Esta es uno de los atributos más importantes de la clase, ya que es aquí donde se guarda la información acerca de los últimos pasos de seguimiento.

  Este atributo no se debe modificar en ningún momento, ya que cualquier modificación puede provocar el fracaso del siguiente paso de seguimiento.

- ippbuf *pbuf
  Guarda la información de la imagen que se va a procesar en el paso presente de seguimiento. La información que contiene es irrelevante de un paso a otro de seguimiento. Su presencia como atributo es para evitar tener que reservar memoria en cada una de las llamadas a los procedimientos de la clase. Se crea en el constructor de la clase y se libera en el destructor, a no ser que se modifiquen las características de la imagen en un nuevo proceso de inicialización de la clase, en cuyo case se libera si el nuevo tamaño de la imagen no coincide con el anterior.

- bool FileIn
  Su valor se actualiza solo en el constructor. Puede tomar los siguientes valores:

  - "true". Si las imágenes a procesar son procedentes de un archivo de arrastre.

  - "false". Si las imágenes a procesar no proceden de un archivo de arrastre, sino que se le van pasando una a una en cada una de las llamadas a los métodos de la clase.

- bool Error
  Esta variable se pone a "true" cuando se ha producido un error interno en el proceso de seguimiento. Mientras esté a "true" la clase no podrá usar ningunos de sus métodos salvo el de inicialización.

  Una vez que la variable se pone a "true" no se puede hacer cambiar su valor a nos ser que se use el método de inicialización de la clase o se elimine el objeto.

- bool ConsPorDefecto
  Si está a "true" indica que se ha utilizado el constructor por defecto para crear el objeto.

La clase posee dos posibles constructores, uno por defecto y otro al cual se le pasa parámetros acerca de la configuración del seguimiento. Los vemos a continuación:

- CTracking()
  Este es el constructor por defecto. Este constructor inicializa todos los parámetros necesarios de la clase. Los parámetros de la configuración del seguimiento son tomados de modo que sean los más genéricos posible. Estos parámetros de configuración del seguimiento pueden ser adecuados para secuencias de imágenes visuales con contraste suficiente. Los parámetros son los que a continuación se detallan:

```
ptrk->fil_in = 0;
ptrk->col_in = 0;
ptrk->anchov = 7;
ptrk->altov = 7;
ptrk->traza = 1;
ptrk->display = 1;
ptrk->filtrar = 1;
ptrk->anchof = 2;
ptrk->altof = 2;
ptrk->sigma = (float)0.7;
ptrk->ver_sel = 2;
ptrk->nivel_av = 10;
ptrk->solape = 7;
ptrk->nv_max = 180;
ptrk->si_presel = 1;
ptrk->ver_trk = 6;
ptrk->corr_norm = 1;
ptrk->ver_corr = 4;
ptrk->error_in = (float)1.7;
ptrk->error_fin = (float)0.7;
ptrk->radio = 100;
ptrk->niter_max = 15;
ptrk->d_conv = (float)0.0100000;
ptrk->tam_cls = 4;
ptrk->radio_cls = 64;
ptrk->precic_cls = (float)0.05;
```

```
ptrk->radio_rep = 35;
ptrk->par_cam.foco = (float)393.3599854;
ptrk->par_cam.kappa = 0;
ptrk->par_cam.escala_x = (float)-1.1764710;
ptrk->par_cam.escala_y = 1;
ptrk->par_cam.cx = 148;
ptrk->par_cam.cy = 128;
pbuf=NULL;
plsec = NULL;
LongHistory = 6;
FileIn=false;
ConsPorDefecto=true;
```

Si se pretende analizar secuencias de imágenes más complejas, como por ejemplo secuencias de imágenes infrarrojas será necesario usar el otro constructor de la clase, el cual permite el paso de parámetros mediante archivo.

Este constructor inicializa la clase para que tome las imágenes una a una, no de archivo, para configurar la toma de imágenes desde archivo arrastre será necesario el uso del otro constructor de la clases.

- CTracking(char *param_file,int long_history)
  A este constructor le podemos pasar como parámetros la ruta de un archivo de parámetros desde el cual cargar los parámetros de configuración del seguimiento y el tamaño de la lista de secuencias.

  En el fichero de parámetros se puede indicar si se desea cargar las imágenes desde un archivo de arrastre o pasarlas una a una, además de especificar la configuración del seguimiento.

  El valor de la longitud de la lista de secuencias debe ser un número estrictamente mayor de 0, en caso contrario la clase asume como tamaño por defecto 6, dando la advertencia correspondiente en la ventana de comandos

- ~CTracking()
  El destructor se encarga de liberar toda la memoria reservada en la clase.

Los métodos que se han implementado en la clase han pretendido ser lo menos complejos posible, de modo que el usuario de la clase no tuviera que conocer los entresijos del funcionamiento del seguimiento. Los métodos son los siguientes:

- void InitTracking(CByteImage *img=NULL)

Este método permite inicializar el seguimiento. Recibe como parámetro una imagen dada por el usuario sobre la cual realizar el primer paso de seguimiento. Debido a que la clase CByteImage es autocontenida, todos los datos acerca de la imagen van dentro de ella.

Si el objeto creado de la clase CTracking está configurado para recibir imágenes desde archivo de arrastre, no es necesario que se le pase ningún argumento a la función.

Esta función se puede llamar tantas veces como se quiera, pero en cada una de las llamadas se inicializará el proceso de seguimiento de imágenes, luego lo normal es llamarla con la primera imagen de la secuencia y volverla a llamar solo para procesar otra secuencia de imágenes o bien como respuesta a un error.

Como dijimos anteriormente, cuando se produce un error, la única forma de desactivar la bandera de error es inicializando la clase mediante este procedimiento.

La función esta protegida frente a posibles errores del usuario, como por ejemplo pasarle imágenes cuando se ha configurado para leer imágenes desde archivo de arrastre. Cuando se producen errores son notificados al usuario mediante una excepción en la ventana de comandos.

Es de señalar que el método de seguimiento que veremos a continuación solo se puede ejecutar si el objeto ha sido inicializado mediante esta función.

- void Tracking(CByteImage *img=NULL)
  El método presente permite realizar un paso de seguimiento sobre la imagen que se le pasa como parámetro.

  Tal y como se dijo antes, no se puede usar este método si no se ha inicializado previamente la clase mediante el método InitTracking(...). En caso contrario se dará una excepción por pantalla.

  Si el objeto está configurado para trabajar con imágenes de arrastre no se le deberá mandar ningún parámetro al método.

  Una vez que la función ha terminado, guarda toda la información de seguimiento en la lista de regiones de modo que el usuario pueda acceder a estos datos después de haber hecho el paso de seguimiento.

- CList<REGION,DOUBLE,LINEAR> *GetRegionList()

Este método nos devuelve un puntero a los últimos datos obtenidos del seguimiento. Es necesario tener en cuenta que no es conveniente que la lista de regiones que se devuelve sea alterada, ya que puede producir errores a la hora de hacer el siguiente paso de seguimiento.

Cada vez que se hace un paso nuevo de seguimiento los datos de la lista de regiones son actualizados, de modo que se pierden. Si el usuario quiere tener constancia de la información de cada uno de los pasos de seguimiento deberá guardar el mismo los datos después de cada iteración.

- lis_sec *GetHistorySequences()
  Devuelve un puntero a la lista de secuencias de la clase, Esta lista no debe ser modificada bajo ningún concepto.

  Este método fue implementado para dar portabilidad a la clase de cara a poder usar funciones en ANSI C ya implementadas que reciben como parámetro esta lista.

  En la medida de lo posible, se pretende que en versiones posteriores de la clase CTracking este método desaparezca para darle a la clase una mayor transparencia.

- void PrintStatistics()
  Este método nos permite imprimir por pantalla y en archivo datos estadísticos sobre el seguimiento realizado.

4.3. Clase CHomography.

Esta clase nos va a permitir implementar las funciones necesarias para poder obtener la homografía a partir de los datos del seguimiento.

Al igual que hicimos antes con la clase CTracking, en este apartado primero estudiaremos las funciones ANSI C en las que se basa y posteriormente se verá el armazón de C++ que se ha implementado para poder dar soporte a las clases.

4.3.1. Funciones ANSI C de la clase CHomography.

A diferencia del seguimiento, las funciones a las que hay que llamar para poder obtener la matriz de homografía que mejor se adapta a los datos de correspondencias que se obtienen del seguimiento son solo una, esta función es Compensa(...). Así lo único que hay que hacer es llamar a esta función cuando se quiera la matriz de homografía, pasándole los datos adecuados.

Veamos a continuación una leve descripción de los tipos de datos y funciones usadas:

- Correspondencia
  Esta es una estructura donde guardamos los datos básicos de una correspondencia. Estos datos son los siguientes:

int NumeroSecuencia;	Numero de secuencia
double FilaActual;	Fila actual de la correspondencia
double ColActual;	Columna actual de la correspondencia
double FilaPrevia;	Fila anterior de la correspondencia
double ColPrevia;	Columna anterior de la correspondencia
int EstadoCorres;	Estado de la correspondencia

  Vemos como estos datos son básicamente la información acerca de la posición anterior y actual de la ventana. Un dato muy importante es EstadoCorres, el cual nos dice si la correspondencia actualmente es un outlier, es válida o no válida. En función del tipo de la correspondencia se le dará un trato u otro a los datos.

- CompensationDatas
  Esta estructura viajará a lo largo de las diferentes funciones en las que se estructura el proceso llevando la información de los errores producidos, los outliers detectados, las correspondencias correctas y otros datos de interés para el procesamiento.

  Los campos de esta estructura son los siguientes:

tipo_error error;	Indica error al realizar la compensación.
double h[9];	Matriz de compensación.
double fiabilidad;	Indica la fiabilidad de la matriz obtenida La fiabilidad máxima es 1 y la mínima 0
int numero_ventanas;	Especifica la longitud de la lista de ventanas
int n_cor;	Numero de correspondencias correctas

int n_cor_lmeds;	Número de correspondencias que, siendo correctas, se van a usar para hallar H mediante el proceso de LMedS
int n_outliers;	Numero de outliers
int *seq_estado_vent;	Puntero a una tabla donde se encuentra caracterizada cada ventana según sea una correspondencia correcta, incorrecta u outlier en la imagen anterior

De todos estos parámetros es conveniente señalar la secuencia de estado de las ventanas. En esta tabla encontraremos una variable por correspondencia que nos indicará si esta es un outlier o no.

Cabe destacar también h, que es la zona donde guardamos la matriz de homografía. En ella está guardada la matriz de homografía de la iteración anterior del algoritmo, ya que esta es indispensable para poder hallar la matriz de homografía respecto a la primera imagen de la secuencia.

- par_comp_mov
  Mediante esta estructura se le dan instrucciones al algoritmo que calcula la matriz de homografía. Podemos considerar esta estructura como una variable de configuración del algoritmo que obtiene la matriz de homografía.

  Los campos de los que está compuesto son los siguientes:

int dim_vec_param;	Número de correspondencias en LMedS
float fiab_corr;	Pb. de error en LMedS
float porc_outliers;	Porcentaje de outliers
ver_warping_t ver_warping;	Versión de compensación
int traza;	Nivel de traza de error
int min_corres;	Número mínimo de correspondencias

  Es de señalar que el parámetro ver_warping está descatalogado y que el valor que se le ponga no afecta para nada al proceso de obtención de la matriz de homografía. Por otro lado es necesario asignarle los valores al resto de parámetros con mucho cuidado, ya que pequeñas variaciones pueden dar lugar a grandes errores.

  Explicaremos el resto de parámetros para una mejor comprensión:

  - dim_vec_param

Indica cual es el número de correspondencias en las que se van a agrupar los subconjuntos del algoritmo de LMedS. Su valor típico es cuatro. No se deben dar valores por debajo de 4, ya que en ese caso la matriz de homografía resultante no es correcta.

- fiab_corr
  Indica la probabilidad con la que se quiere que no haya outliers después del algoritmo de LMedS. Generalmente se le suele dar un 99% de probabilidades.

- porc_outliers
  Especifica el porcentaje de outliers que encontraremos entre los datos de entrada al LMesS. Generalmente se suele tomar un 40%.

- min_corres
  Cuando el algoritmo no posee suficientes correspondencias no se comporta de forma correcta debido a la falta de información. Es por esto por lo que es necesario imponer un número mínimo de correspondencias, de modo que si recibe menos de este número de un error.

- void calcula_corres(lis_sec *plist,CompensationDatas *compensacion,Correspondencia *TablaCorres)
  Esta función nos permite saber cuales de las correspondencias que se reciben del seguimiento son válidas y cuales no lo son.

  Para esto se diseño una función con la idea de poder actualizarla después si fuera necesario en función de las modificaciones del seguimiento. Además nos permite cambiarla en caso de no recibir una lista de secuencias, sino una lista de regiones.

  La función recibe como parámetro:

  - La lista de secuencias que contiene la información de la evolución de las correspondencias de una iteración a otra.

  - Una estructura del tipo CompensationDatas donde se encuentra la información acerca del estado actual del sistema de obtención de matriz de homografía.

  - Una tabla de Correspondencia donde iremos metiendo la información de cada una de las correspondencias obtenidas de la lista de secuencias.

Esta función lleva incorporada la dinámica adecuada para poder elegir ente los puntos que posteriormente se usarán en el algoritmo de LMedS. Recordemos que para obtener cuales eran dichos puntos era necesario seguir lo siguiente:

- Si la correspondencia es incorrecta ➔ Correspondencia incorrecta.
- Si la correspondencia correcta y nueva ➔ Correspondencia adecuada para M-Estimaciones.
- Si la correspondencia es correcta y fue outlier en la iteración pasada ➔ Correspondencia adecuada para M-Estimaciones.
- Si la correspondencia es correcta y no fue outlier en la iteración anterior ➔ Correspondencia adecuada para LMedS.

- void compensa(Correspondencia *TablaCorres,par_comp_mov *ppar,CompensationDatas *compensacion,int ancho, int alto)
  Esta función es la que nos va a facilitar el valor de la matriz de homografía, es la que implementará todos los métodos vistos en puntos anteriores.

  La función recibe los siguientes parámetros:

  - Tabla de Correspondencia donde está la información de todas las correspondencias recibidas del seguimiento.
  - Los parámetros de configuración del algoritmo.
  - Una estructura on los datos de compensación necesarios, tales como la matriz de homografía en la iteración anterior.
  - El ancho de la imagen, necesario para poder definir las rejillas de la imagen.
  - El alto de la imagen, necesario para poder definir los bucket de la imagen.

  La matriz de homografía resultante se devuelve en el campo h de la estructura CompensationDatas que se le pasa como parámetro.

4.3.2. Métodos y atributos de la clase CHomography.

Empezaremos definiendo cuales son los atributos de la clase:

- double H[9]
  En esta matriz guardamos la homografía en cada iteración.

  Su contenido se actualiza cada vez que se lleva a cabo la obtención de la matriz de homografía, luego es necesario que el usuario la guarde si desea tener un almacenamiento de las matrices obtenidas en cada iteración.

  El usuario puede modificar su contenido todas las veces que quiera ya que en cada iteración se guardan los datos de homografía variables internas a la clase.

- bool InternalError
  Esta variable binaria puede tomar los siguientes valores:

  - "true". Si ha ocurrido un error interno durante cualquiera de los procedimientos de la clase.

  - "false". Si no ha habido ningún tipo de error.

  Cuando se produce un error interno no se puede ejecutar ninguno de los métodos mientras que no se de un reset al objeto mediante el procedimiento adecuado.

- int ImageWidth
  Esta variable entera guarda el ancho de la imagen sobre la cual se ha aplicado el seguimiento de ventanas.

  El valor de esta variable no se puede cambiar a no ser que inicialicemos el objeto, ya que de ella depende el buen funcionamiento de la detección de outliers en el algoritmo de LMedS.

- int ImageHeight
  Esta variable entera guarda el alto de la imagen sobre la cual se ha aplicado el seguimiento de ventanas.

  El valor de esta variable no se puede cambiar a no ser que inicialicemos el objeto, ya que de ella depende el buen funcionamiento de la detección de outliers en el algoritmo de LMedS.

- int RegionsNumber
  Esta variable contiene el número de regiones de las que está compuesta la lista de secuencias que se le pasa como parámetro a la clase.

  Al igual que antes, esta variable no puede ser modificada en plena ejecución de la estabilización de una secuencia, ya que haríamos que el programa no funcionara correctamente.

  Para poder cambiar su valor es necesario inicializar de nuevo el objeto.

- CompensationDatas *CompData
  CompData contiene los datos sobre la compensación en iteraciones anteriores del algoritmo, de modo que contiene la matriz de homografía anterior y otros datos. Esta variable no puede ser modificada en durante la estabilización de una secuencia de imágenes.

- par_comp_mov Parameters
  Contiene los datos relativos a la configuración del algoritmo de cálculo de matriz de homografía.

  La configuración de la función de compensación está fijada en los procedimientos, de modo que el usuario no tiene porque percibir la complejidad que esto significa.

En los puntos sucesivos vamos a definir la funcionalidad del constructor y destructor de la clase:

- CHomography()
  Este constructor se encarga de inicializar los atributos de la clase a valores conocidos, reservando la memoria necesaria.

- ~CHomography()
  El destructor libera la memoria que se halla reservado el constructor o durante algunos de los procedimientos de la clase.

Por último veremos cuales son los procedimientos de la clase y como debemos hacer las llamadas a cada uno de ellos:

- bool SetParameters(int RegionsNumber, int MinCorrespondences, int ImgWidth, int ImgHeight)
  Este procedimiento permite ajustar los parámetros de la clase. Podemos ver como no recibe todos los parámetros que aparecen en la estructura de

configuración del algoritmo de obtención de matriz de homografía. Esto es debido a que hay parámetros que los configura automáticamente, tales como el número de correspondencias de cada subconjunto en el algoritmo de LMedS.

La clase lleva incorporadas funciones que permiten detectar si los parámetros son correctos, de modo que, por ejemplo, el número de correspondencias debe ser un entero mayor que 5.

Al mismo tiempo, el valor de la anchura y la altura deben ser enteros mayores o iguales a 1 y el número mínimo de correspondencias no puede ser superior al número de correspondencias totales.

Si se produce algún error la función devolverá "false" y se activará el atributo de error interno, el cual solo se podrá desactivar si se da un reset al objeto mediante el procedimiento que veremos a continuación.

- void Reset()
  Este procedimiento nos permite dar un reset al objeto, de modo que podamos partir de cero en lo que a la matriz de homografía se refiere.

  Es necesario aplicar un Reset cuando se han producido errores internos en el procesamiento de algunos de los métodos de la clase. Al mismo tiempo el Reset nos permite reinicializar los parámetros que se asignan en el constructor, de esta manera podemos reutilizar un objeto, sin necesidad de destruirlo.

  También es necesario aplicar un Reset cuando el método que nos permite obtener la matriz de homografía nos devuelve un error, ya que este tipo de errores también se considere error interno.

  Tenemos que tener en cuanta que cada vez que se hace un reset se pierde toda la información de las posiciones anteriores, de modo que la matriz de homografía se referencia respecto de la nueva imagen que se reciba. Así perderemos toda referencia de posición cuando demos un reset al objeto.

- HomographyErrors ComputeHomography(lis_sec *Data)
  Este es el método principal de la clase. Nos permite obtener la matriz de homografía que mejor se ajusta a los datos que se le pasan como parámetros.

  Usando las funciones de ANSI C que hemos visto anteriormente, este método hace la conversión de los datos que recibe a una tabla de tipo Correspondencia para poder procesarlos, posteriormente llama a la función Compensa(...).

Vemos que la función devuelve un código de error en caso de haberlo. Los posibles códigos de error son los siguientes:

- NO_ERRORS
  Esta etiqueta indica que no ha habido ningún error en el procesamiento de la matriz de homografía.

- INSUFFICIENT_CORRESPONDENCES
  Este error se da cuando el número de correspondencias recibidas para obtener la matriz de homografía es inferior al mínimo establecido por el usuario.

  En este caso el algoritmo deja de procesar y lanza una excepción, pasando un mensaje de aviso por pantalla. El usuario debe decidir si desea continuar con el funcionamiento del programa o desea abortar.

  En el caso de querer continuar procesando será necesario que el usuario use el método de Reset(...) ya que se ha producido un error interno.

- TOO_MANY_OUTLIERS
  Indica que en el procesamiento del algoritmo se han detectado demasiados outliers, más del 40% del total de las correspondencias, lo cual quiere decir que los datos de entrada no son admisibles por el algoritmo.

  En este caso, al igual que antes, se producirá un error interno que se indicará al usuario, el cual será el encargado de manejarlo, debiendo dar un reset si quiere seguir usando el objeto.

- CLASS_NOT_INITIALIZED
  La estructura interna de la clase es tal que no se puede usar este método a no ser que se haya usado previamente el método de inicialización de parámetros internos de la clase.

  Si un usuario ejecuta este método sin haber inicializado los parámetros de se volverá este error y será necesario resetear el objeto.

- INTERNAL_ERROR
  El método devuelve este error cuando se producen errores no controlados en la función Compensa(...), tales como la resolución de sistemas de ecuaciones incompatibles, errores en la reserva dinámica de memoria.

En algunos de estos errores se pasará información por pantalla para indicar la naturaleza del error, pero en otros mucho menos esperados, no habrá información.

Al igual que en todos los casos anteriores, es necesario resetear el objeto para poder seguir usándolo.

Después de ejecutar este método se actualizarán los datos acerca de la homografía, los outliers y las correspondencias. Además, se copiará la nueva matriz de homografía en el atributo H para que el usuario pueda acceder a los datos obtenidos.

- double *GetHomography()
  Mediante este método podemos tener acceso a la matriz de homografía que se ha obtenido en el último paso de procesamiento de la misma.

  Como adelantábamos anteriormente, esta función devuelve un putero a un atributo de la clase que puede ser modificado por el usuario sin ningún tipo de reservas, ya que no influye en el procesamiento de la siguiente matriz de homografía, los datos relevantes están guardados en atributos a los que el usuario no tiene acceso.

  Si se ha producido un error interno no se podrá tener acceso a los datos hasta que no resetear el objeto.

4.4. Clase CWarping.

Una vez extraída la matriz de homografía que mejor se ajusta a los datos de correspondencias obtenidos del seguimiento usaremos esta para poder rotar, trasladar y deformar la imagen de modo que se ajuste lo mejor posible a la imagen de anterior.

Como peculiaridad veremos que en este caso no hay funciones en ANSI C sobre las que basarnos. Debido a que en ANSI C era necesario recibir punteros a la imagen compensada constantemente y el usuario debía de encargarse del mantenimiento de la misma, se decidió encapsularlo directamente en C++, ya que a fin de cuentas la compensación en si es tan solo una función. Veremos a continuación los atributos y métodos de esta clase.

Los atributos de los que está compuesta la clase son los que a continuación se detallan:

- ImageTypes IType
  Este enumerado contiene información acerca del tipo de imagen que se espera recibir para el proceso de compensación.

  Podemos recibir tres tipos de imágenes, cada una de ellas con su propio identificador. Los tipos son los siguientes:

  - B_W
    Imagen en blanco y negro. Cada píxel está determinado por un octeto.

  - RGB
    Imagen en color, cada píxel está formado por tres octetos, uno para cada componente básico del color.

  - BOOLEAN
    Todos sus píxeles son de tipo bool, de modo que la imagen resultante es binaria.

  Cuando se configura el objeto, en IType se guarda información ha cerca del tipo de imagen para la cual ha sido configurado el objeto. Si en algún momento se le pasa un tipo diferente al especificado se generará un error y se pasará un mensaje por pantalla.

- WarpingTypes WType
  Este enumerado contiene el tipo de compensación que se va a implementar en los métodos del objeto.

  Podemos encontrar dos tipos etiqueta:

  - BILINEAL
    Esta etiqueta indica que la compensación que se va a realizar está basado en interpolación bilineal.

  - MORE_SIMILAR_PIXEL
    Esta etiqueta indica que la compensación está configurado para que realice una interpolación de la imagen según el píxel más parecido.

  Esta variable guardará la configuración del tipo de la compensación para el cuál está configurado el objeto.

- ROI roi

roi es un objeto de la clase ROI. Este objeto nos permite especificar regiones de interés, indicando el punto de partida dentro de la imagen, el punto final y el paso de downsample realizado sobre la misma, para las filas y columnas.

Esta roi nos va a permitir realizar la compensación solo de la zona deseada de la imagen, no de toda. Es de señalar que el parámetro que indica el downsample es ignorado en la clase CWarping, ya que, por el momento, la clase no lo soporta.

Los métodos de la clase realizarán solo la compensación a la zona de la imagen que se le indique mediante esta variable, a la cual se le da valor mediante el método de actualizar parámetros.

- double Quality
  Esta variable guarda la calidad de la transformación indicada por el usuario de la clase.

  Esta variable puede tomar valores entre 0 y 1, siendo 1 la mayor calidad alcanzable por la clase y 0 la menor posible. La calidad habrá que ajustarla en función del tipo de la dificultad en realizar la compensación.

  Es necesario apuntar que a mayor calidad mayor será el tiempo necesario para poder hacer la compensación de la imagen. Un valor de 0.5 aporta buenos resultados en término general.

- CByteImage *ByteImage, *OutByte
  Este atributo contiene dos punteros a clases del tipo CByteImage para almacenar la imagen compensada e imagen devuelta por la clase cuando la clase está configurada para trabajar con imágenes en blanco y negro.

- CRGBImage *RGBImage, *OutRGB
  Este atributo contiene dos punteros a clases del tipo CRGBImage para almacenar la imagen compensada e imagen devuelta por la clase cuando la clase está configurada para trabajar con imágenes en RGB.

- CBooleanImage *BooleanImage, *OutBoolean
  Este atributo contiene dos punteros a clases del tipo CBooleanImage para almacenar la imagen compensada e imagen devuelta por la clase cuando la clase está configurada para trabajar con imágenes en booleanas.

- bool InternalError
  Esta variable indica que ha habido un error interno en cualquiera de los métodos de la clase.

Una vez que esta variable se activa, la única forma de ponerla a "false" es mediante el procedimiento de reset(...).

Veremos a continuación las tarea realizadas por el constructor y el destructor de la clase:

- CWarping()
  Este constructor nos va a permitir inicializar todos los atributos de la clase de modo que se pueda saber en todo momento que variables han sido utilizadas por la clase, de modo que el destructor pueda identificar cuales son las variables que debe liberar.

  No es suficiente con crear el objeto para poder usar el método que realiza la compensación sobre la imagen que se le pasa, es necesario especificar los parámetros, ya que el constructor tan solo les da valores meramente identificativos.

- ~CWarping()
  El destructor se encarga de eliminar toda la memoria reservada para el objeto, de modo que el usuario tan solo debe trabajar con punteros a datos, no tiene que molestarse en reservas dinámicas ni en liberaciones de las mismas.

Por último veremos cuales son los métodos que se han implementado en la clase para realizar la compensación:

- bool SetParameters(ROI roi, ImageTypes IType, WarpingTypes WType, double Quality)
  Este método nos permite dar valores a los parámetros de la clase. Lo que significa cada uno de las variables está explicado en su correspondiente atributo en secciones anteriores.

  Lo primero que hace la función es verificar si los valores que se le pasan como parámetros son correctos, realizando las siguientes comprobaciones:

  - Comprueba que roi.xini>0 y roi.yini>0, ya que en caso contrario nos saldríamos fuera de la imagen. No se comprueba roi.xend y roi.yend debido a que aun no disponemos información sobre el tamaño de la imagen.

  - Comprobamos que el valor de la calidad está comprendido entre 0 y 1.

- Comprobamos que el tipo de compensación que se especifica lo soporta la clase.

- Comprobamos que el tipo de imagen que se especifica es soportado por la clase.

Es necesario tener en cuenta que el parámetro del tipo de compensación es ignorado cuando la imagen a la que se aplica es boolena, ya que estas imágenes solo poseen un tipo de compensación, el cual no es ni bilineal y de píxel más parecido.

- void Reset()
  Este método permite resetear el objeto al completo, dejándolo en la mimas condiciones que si lo hubiera creado con el constructor de la clase.

  Es necesario recalcar que cuando se da un reset toda la información acerca de la imagen compensada se pierde. Por otro lado, el reset es la única manera de hacer que el objeto se recupere de un error interno.

  Cuando se da un reset se inicializan dotas las variables y se libera toda la memoria reservada hasta el momento.

- CByteImage *ComputeWarping(CByteImage *Image, double *h=NULL)
  La presente función realiza la compensación de la imagen en blanco y negro que se le para como parámetro usando la matriz de homografía que le acompaña.

  La función tendrá dos modos de funcionamiento:

  - Se le pasa una imagen y una matriz de homografía. En este caso realiza la compensación de movimiento, guardando la imagen compensada en variables internas.

  - Se le pasa solo una imagen. En este caso, la función supone que la imagen que se le está pasando es la primera de la secuencia, de modo que la almacena para posteriores pasos en la compensación de la secuencia y devuelve el control.

  En ambos casos se hace una copia de la imagen compensada para que el usuario pueda usarla y modificarla si es necesario, pero nunca liberarla.

  En función del tipo de compensación elegido se hará una aproximación bilineal o del píxel más próximo.

- CRGBImage *ComputeWarping(CRGBImage *Image, double *h=NULL)
  La presente función realiza la compensación de la imagen en color que se le para como parámetro usando la matriz de homografía que le acompaña.

  La función tendrá dos modos de funcionamiento:

  - Se le pasa una imagen y una matriz de homografía. En este caso realiza la compensación de movimiento, guardando la imagen compensada en variables internas.

  - Se le pasa solo una imagen. En este caso, la función supone que la imagen que se le está pasando es la primera de la secuencia, de modo que la almacena para posteriores pasos en la compensación de la secuencia y devuelve el control.

  En ambos casos se hace una copia de la imagen compensada para que el usuario pueda usarla y modificarla si es necesario, pero nunca liberarla.

- CBooleanImage *ComputeWarping(CBooleanImage *Image, double *h=NULL)
  La presente función realiza la compensación de la imagen binaria que se le para como parámetro usando la matriz de homografía que le acompaña.

  La función tendrá dos modos de funcionamiento:

  - Se le pasa una imagen y una matriz de homografía. En este caso realiza la compensación de movimiento, guardando la imagen compensada en variables internas.

  - Se le pasa solo una imagen. En este caso, la función supone que la imagen que se le está pasando es la primera de la secuencia, de modo que la almacena para posteriores pasos en la compensación de la secuencia y devuelve el control.

  En ambos casos se hace una copia de la imagen compensada para que el usuario pueda usarla y modificarla si es necesario, pero nunca liberarla.

Capítulo 5

Posibles ampliaciones

5.1. Introducción.

A lo largo de la memoria se ha hecho referencia a posibles ampliaciones relacionadas con diferentes aspectos del proyecto. En este punto vamos a recopilarlas todas y añadir nuevas, especificando en que consiste cada una de ellas y las razones por las que sería interesante llevarlas a cabo.

Como veremos a continuación, algunas de estas posibles ampliaciones ya están totalmente desarrolladas y lo único que habría que hacer sería codificarlas para poder hacer uso de ellas.

5.2. Normalización de los en el algoritmo de M-Estimaciones.

Si se hace un estudio de cuales son las características de los datos que le llegan al algoritmo de M-Estimaciones se comprobará que la disparidad de los datos que llega a este algoritmo es bastante elevada.

Dicha disparidad en los datos para los cuales el algoritmo de M-Estimaciones debe encontrar un modelo común es debida a que las correspondencias que usamos para saber cual es el movimiento de una imagen a la siguiente pueden estar repartidas por toda la imagen.

Si suponemos una imagen de tamaño (alto, ancho), las correspondencias que recibe el algoritmo pueden estar referenciadas a píxeles que van desde el (0,0) hasta el (alto, ancho), de modo que la disparidad de valores es clara.

Esta disparidad en los datos de entrada genera que la solución del sistema de ecuaciones resultante menos exacta ya que los pesos y distancias de los datos se hacen muy diferentes en función de las características de estos. Veamos un ejemplo:

> Sea A=10000.1 y B=10000, si las restamos obtendremos que el error en valor absoluto es 0.1, el cual se puede considerar pequeño dado el enorme valor de las cifras que estamos restando.
>
> Por otro lado si A=0.7 y B=0.8, el error en valor absoluto sigue siendo el mismo, pero en este caso si es un error significativo, ya que es del mismo orden de las cifras que estamos restando.
>
> Si nuestro algoritmo debe establecer que todo error inferior a una determinada cota debe ser despreciable, vemos como dicha cota depende directamente de la magnitud del dato que estemos tratando.

Lo ideal sería aplicar algún tipo de transformación que permitiera homogeneizar los valores de los datos, de modo que en media todos los elementos estén a la misma distancia del origen, minimizando de esta manera este efecto de disparidad de los datos y dando lugar sistemas de ecuaciones cuya resolución lleva menor índice de error intrínseco.

En [1] encontrar una forma de proceder para poder normalizar los datos. El método que se explica no es demasiado complejo y podría ser implementado con facilidad.

5.3. Compensación sobre imágenes en espacio homogéneo.

En apartados anteriores se hizo referencia a que la compensación de la imagen en un espacio que fuera homogéneo generaría una mejor transformación de la imagen, ya que de esta manera las distancias entre los colores de los píxeles coincidirían con la percepción del ser humano.

Para poder implementar esta transformación no habría que alterar el algoritmo que ya esta implementado, bastaría con realizar la transformación al principio del mismo, midiendo las distancias en la imagen transformada y tomando los píxeles de la imagen en el espacio RGB.

La aplicación de este método generaría probablemente una mejora en la calidad de la imagen estabilizada.

5.3. Compensación con ajuste automático del número de segmentos de linealización.

Anteriormente vimos el estudio acerca de las posibilidades de linealización de la transformación mediante homografía. Se llegó a la deducción que sería interesante implementar algún algoritmo que definiera de forma automática el número de tramos de linealización necesarios para obtener una buena aproximación de la transformación.

Igualmente se hicieron pruebas que nos permitieron establecer diversos valores del número de tramos en función del valor máximo entre H[6] y H[7] de la matriz de homografía. Con los valores obtenidos resulta fácil obtener la curva que mejor se ajusta a los datos, obteniéndose así una ecuación que, sin más que sustituir el valor máximo de entre H[6] y H[7] nos dé el número adecuado de tramos de linealización.

Posteriormente bastaría con definir un número máximo de tramos de linealización en función del tamaño de la imagen a tratar para no incurrir en el error de generar un número de tramos tan grande que genere un mayor tiempo de computación que la transformación sin linealizar.

5.4. Submuestreo en seguimiento y compensación.

Cuando tratamos con imágenes muy grandes nos podría resultar de gran interés que el algoritmo de seguimiento nos permitiera realizar un submuestreo de la imagen y que solo se trabajara en el seguimiento con esta imagen reducida. De esta manera conseguiríamos disminuir notablemente el tiempo de computación del algoritmo.

Hay que tener en cuenta que las funciones ANSI C sobre las que se apoya la clase CTracking solo permiten seleccionar una zona de la imagen, pero no seleccionar una zona con submuestreo, de modo que esto lo tendríamos que hacer a un nivel superior, dentro de la clase y sin que el usuario se percatara de ello.

Por otro lado, al igual que la clase CTracking, a la clase CWarping se le puede especificar una zona concreta sobre la que hacer la compensación, pero esta zona no puede tener submuestreo. También nos podría ser de gran ayuda tener a nuestra disposición esta herramienta ya que, al igual que antes, nos ahorraríamos un tiempo considerable a la hora de calcular la compensación de la imagen.

Estas modificaciones no son demasiado complejas de llevar a cabo y se podrían hacer sin más que dedicarle algo más de 2 semanas a codificar las modificaciones. A cambio obtendríamos clases más versátiles y con una mayor capacidad de adaptación a las circunstancias de la imagen.

Capítulo 6

Bibliografía

[1] Richard I. Hartley, "In defense of the height-point algorithm", IEEE transactions on pattern anlysis and machine intelligence, Vol. 19 No. 6 June 1997.

[2] Zhengyou Zhang, "Parameters estimation techniques. A tutorial with application to conic fitting", INRIA, No. 2676 October 1995.

[3] David J. Kruglinski, "Programación avanzada con Microsoft Visual C++", Microsoft Press, McGraw Hill.

[4] Per-Erik Forssén, "Updating camera location and heading using a sparse displacement field", Report LiTH-ISY-R-2318, November 2000