ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA INFORMÁTICA

Detección automática de cambios urbanos a partir de imágenes de satélites

Realizado por

Vicente Manuel Arévalo Espejo

Dirigido por

Javier González Jiménez

Co-Dirigido por

Gregorio Ambrosio Cestero

Departamento

Ingeniería de sistemas y automática

UNIVERSIDAD DE MÁLAGA

MÁLAGA, MARZO DEL 2001

UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA INFORMÁTICA

Reunido el tribunal examinador en el día de la fecha, constituido por:

Presidente D/D^{a}: _____

Secretario D/D^{a}: _____

Vocal D/D^{a}: _____

para juzgar el proyecto Fin de Carrera titulado:

_____

_____

del alumno D/D^{a}: _____

dirigido por D/D^{a}: _____

co-dirigido por D/D^{a}: _____

ACORDÓ POR _____ OTORGAR LA CALIFICACIÓN DE _____ Y PARA QUE CONSTE, SE EXTIENDE FIRMADA POR LOS COMPARECIENTES DEL TRIBUNAL, LA PRESENTE DILIGENCIA.

Málaga, a _____ de _____ de _____

El Presidente	El Secretario	El Vocal
Fdo: _____	Fdo: _____	Fdo: _____

Índice general

Índice de figuras

A mis padres y hermana, y a mi novia.

Capítulo 1

Presentación

1.1 Introducción

El desarrollo de la teledetección ha sido vertiginoso en los últimos años y el futuro inminente parece garantizar un crecimiento incluso más acelerado, si se confirman los múltiples proyectos de nuevos satélites que se encuentran en avanzado estudio. La ampliación de los países generadores de datos, y la irrupción de los consorcios privados, hace prever un aumento exponencial de las imágenes disponibles sobre nuestro planeta. Con el notable avance de los sistemas de comunicación, almacenamiento y análisis digital, el principal cuello de botella que parece vislumbrarse es la falta de aplicaciones capaces de explotar íntegramente este gran volumen de datos.

Una de las aportaciones más destacadas de la teledetección espacial al estudio del medio ambiente es su capacidad para seguir procesos dinámicos. Al tratarse de información adquirida por un sensor situado en una órbita estable y repetitiva, las imágenes de satélite constituyen una fuente valiosísima para estudiar los cambios que se producen en la superficie terrestre, ya sean debidos al ciclo estacional de las cubiertas, ya a catástrofes naturales o a alteraciones de origen humano.

En los últimos años han proliferado notablemente los estudios de detección de cambios, aplicándose a una gran variedad de disciplinas: crecimiento urbano, desecación de humedales, efectos de incendios o plagas, etc.

1.2 Objetivos

El objetivo de este proyecto es el desarrollo de una aplicación que automatice los procesos necesarios para el estudio de series multi-temporales de imágenes con objeto de detectar cambios urbanos.

Los datos proporcionados por esta aplicación deberán ser utilizados por un GIS (Geographic Information System), concretamente, el GIS de la gerencia de urbanismo del Iltmo. Ayuntamiento de Málaga, para determinar la legalidad o ilegalidad de los cambios urbanos detectados.

Figura 1.1: Benalmádena'99.

Figura 1.2: Benalmádena'00.

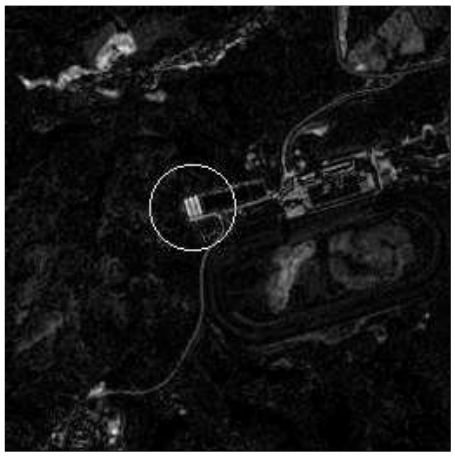


Figura 1.3: Imagen de cambios.

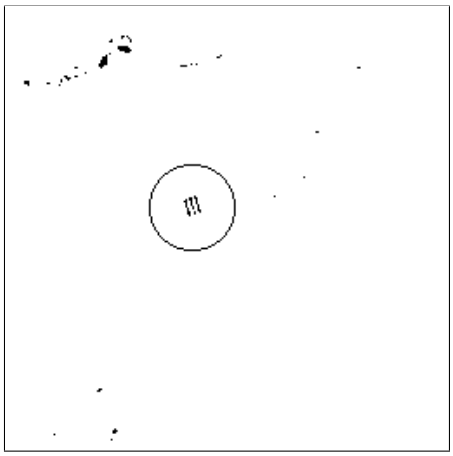


Figura 1.4: Imagen de cambios (t=160).

La aplicación llevará a cabo los siguientes procesos con objeto de alcanzar el objetivo establecido previamente:

- Procesamiento de la serie multi-temporal:
  - Corrección geométrica:
    - *image-to-map* (imagen de referencia)
    - *image-to-image*
  - Corrección radiométrica (o normalización radiométrica)
- Selección y aplicación del algoritmo de detección de cambios apropiado
- Visualización cromática de los cambios detectados

De igual forma, se desarrollará un GUI (Graphic User Interface) multi-plataforma (TCL/TK) con objeto de facilitar al usuario final el uso de la aplicación y la supervisión de los datos (imágenes, histogramas, puntos de interés, etc.) obtenidos.

Los resultados del estudio serán publicados en la WEB, junto con toda aquella información (enlaces de interés, algoritmos, etc.) de la que se ha hecho uso en el desarrollo del proyecto.

1.3 Motivación

La teledetección espacial cuenta con numerosas aplicaciones, gracias a las ventajas que ofrece frente a otros medios de observación más convencionales, como la fotografía aérea o los trabajos de campo, aunque más que sustituirlos los complementa. Entre las ventajas de esta observación espacial, podemos destacar las siguientes:

- Cobertura global y periódica de la superficie terrestre
- Visión panorámica
- Información sobre regiones no visibles del espectro
- Formato digital

Esta relación no implica, naturalmente, que consideremos la teledetección espacial como una panacea para monitorizar, detectar o identificar cambios en la superficie observada. También presenta algunas limitaciones, como son las derivadas de la resolución *espacial*, *espectral* o *temporal* disponibles, que pueden no ser suficientes para ciertos problemas.

1.3.1 Detección de cambios

La detección de cambios a partir de imágenes de satélite se emplea para detectar las modificaciones que ha experimentado un determinado territorio, como consecuencia de un fenómeno natural o de origen humano. La evaluación de los efectos de un incendio forestal o una inundación se incluyen en este ámbito, así como el seguimiento de otros procesos más graduales, como sería la dinámica de cultivos o la detección de cambios urbanos.

La frecuencia con la que se realiza el seguimiento de estos procesos depende principalmente de su dinamismo. La evaluación de los efectos de un incendio forestal o inundación requieren un período corto de tiempo, con objeto de paliar sus consecuencias; los cambios agrícolas precisan una estimación anual, mientras los urbanos pueden hacerse cada seis meses.

En la mayor parte de los casos, la detección de cambios se realiza comparando, píxel a píxel, los niveles de gris de las distintas imágenes. En consecuencia, es necesario eliminar previamente cualquier cambio en los niveles de gris de la escena que no sea debido a cambios reales en la cubierta. Esto implica ajustar con precisión, tanto geométrica, como radiométricamente, las imágenes que intervienen en el análisis, ya sea de forma *absoluta* o *relativa*.

1.3.2 Morfología urbana

Teniendo en cuenta la gran complejidad espacial del fenómeno urbano, en donde conviven actividades muy variadas sobre un reducido espacio, la aplicación de la teledetección espacial a estos ámbitos es reciente y aún limitada. El principal problema, hasta hace bien poco, radicaba en la *resolución espacial* de los sensores, excesivamente groseros para aportar información sobre el entramado urbano. Sin embargo, actualmente, están disponibles comer-

cialmente imágenes con resoluciones espaciales inferiores a un metro[1].

Figura 1.5: Cámara PAN del satélite IRS-1D.

Hablamos, por tanto, de un campo de aplicación incipiente, y con un número potencial de usuarios importante, constituyendo todo un reto el estudio y desarrollo de técnicas (algoritmos, hardware/software, etc.), que unidas a la mejora progresiva de los sensores actuales (ver figura 1.5), permitan un importante y provechoso avance en este prometedor campo.

1.4 Estructura del sistema

A la hora de desarrollar un proyecto como el que nos ocupa se han de tener en cuenta aspectos tales como: el ámbito de aplicación, el sistema sensor, condiciones atmosféricas, etc.

- Ámbito de aplicación
  - Definir el área de estudio
  - Especificar la frecuencia del estudio (estacional, anual, semestral, etc.)

[1]El precio de estas imágenes, comparado con el de las imágenes que utilizamos en este proyecto, es realmente importante, lo que hace aún prohibitiva su utilización. Además, obliga a trabajar con mosaicos de imágenes, lo que dificulta notablemente el proceso de detección de cambios.

- Consideraciones de importancia
  - Sistema sensor:
    - Resolución:
      - Espacial
      - Espectral
      - Radiométrica
    - Frecuencia de cobertura
  - Condiciones atmosféricas:
    - Nubosidad (visibilidad)
    - Condiciones de humedad del suelo
    - Estado de la marea

1.4.1 Etapas

Las etapas a considerar en cualquier proceso de detección digital de cambios son las siguientes:

1. Seleccionar los datos apropiados:
   - Datos colaterales (geográficos, atmosféricos, etc.)
   - Serie multi-temporal de imágenes
2. Procesar la serie multi-temporal de imágenes:
   - Corrección geométrica (registro geométrico)
   - Corrección radiométrica (normalización radiométrica)
3. Seleccionar y aplicar el algoritmo de detección de cambios apropiado
4. Visualizar los cambios detectados (mapa de cambios)

Destaca, por su especial relevancia, la etapa destinada al procesamiento de los datos, concretamente, las técnicas dirigidas a corregir, geométrica y radiométricamente, la serie multi-temporal de imágenes y a detectar cambios en la misma.

Corrección geométrica

Para abordar la detección digital de cambios es preciso que las imágenes se ajusten con gran nivel de detalle, ya que de otro modo estaríamos detectando como cambios lo que sería sólo consecuencia de una falta de ajuste entre imágenes. El efecto puede llegar a ser muy grave, especialmente cuando se trata de detectar cambios en regiones con una gran variabilidad espacial, como es el caso de las urbanas, dependiendo también de la resolución espacial del sensor (ver figuras 1.6 y 1.7).

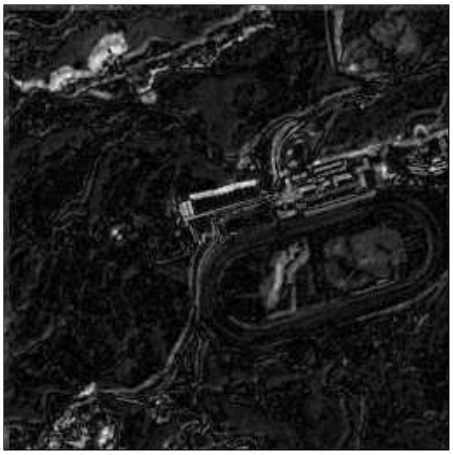


Figura 1.6: Mal ajuste geométrico.

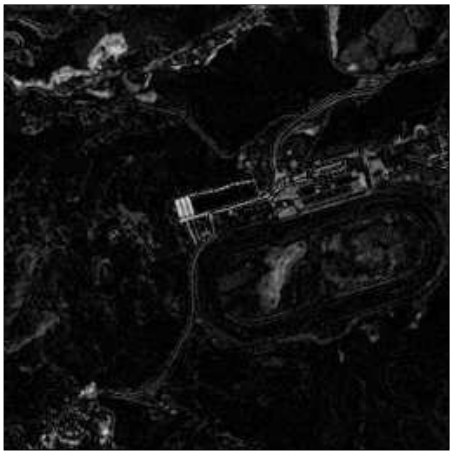


Figura 1.7: Buen ajuste geométrico.

Corrección radiométrica

Otro importante problema en la detección de cambios es el producido por las variables condiciones de observación, situaciones atmosféricas o condiciones de calibración del sensor. Estos efectos modifican la signatura espectral de un píxel, aunque se mantenga constante la cubierta. En consecuencia, es preciso homogeneizar los niveles de gris de las imágenes que intervienen en el análisis (ver figuras 1.8, 1.9 y 1.10).

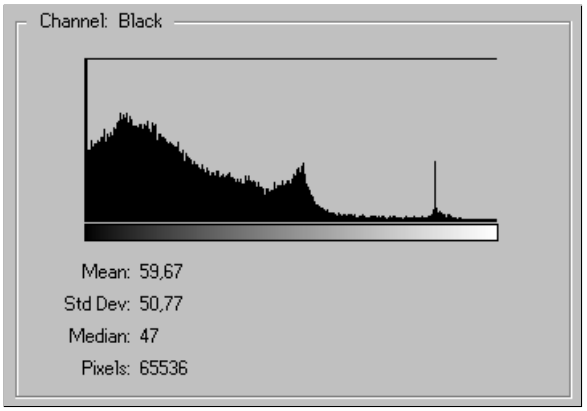


Figura 1.8: Histograma de Benalmádena'99 (figura 1.1).

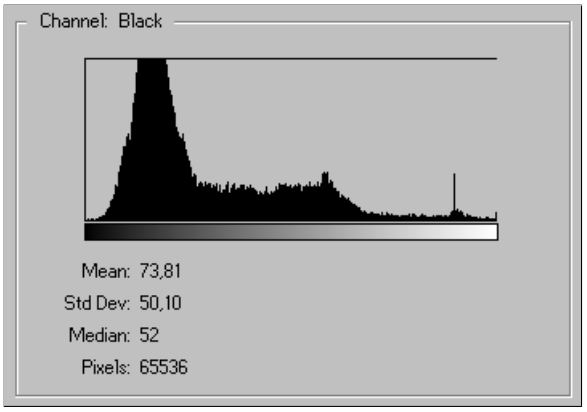


Figura 1.9: Histograma de Benalmádena'00 (figura 1.2).

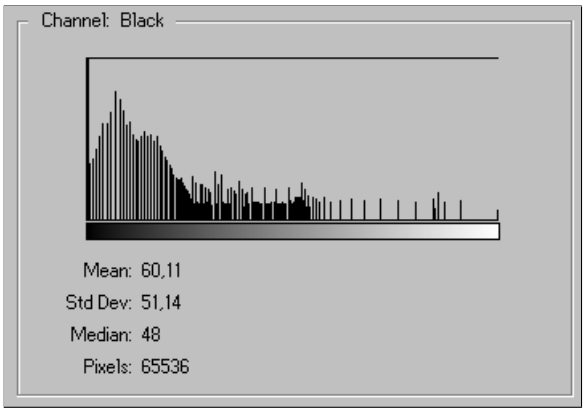


Figura 1.10: Histograma de Benalmádena'00 corregida radiométricamente.

1.5 Recursos

En este capítulo detallaremos cada uno de los recursos que serán empleados en el desarrollo del proyecto, haremos especial hincapié en los siguientes tipos:

- Recursos software
- Recursos hardware
- Datos (imágenes, datos colaterales, etc.)

1.5.1 Recursos software

Esta sección incluye los *recursos software* que serán utilizados en el desarrollo de la aplicación:

Programación

En el desarrollo del proyecto se emplearán las siguientes herramientas de programación:

- Microsoft Visual C++ 6.0
  - Entorno de desarrollo de aplicaciones en C y C++.
- Visual TCL 1.2
  - Entorno de desarrollo de aplicaciones en TCL/TK.

Documentación

En el desarrollo del proyecto se emplearán las siguientes herramientas de documentación:

- WinEdt 5.2
  - Entorno de edición de textos en formato LaTeX.
- LaTeX 1.2e
  - Formateador de textos LaTeX.

Edición WEB

En el desarrollo del proyecto se emplearán las siguientes herramientas de edición WEB:

- Macromedia Dreamweaver 3.0 / Fireworks 3.0
  - Entorno de desarrollo de sitios WEB.
- Adobe Photoshop 4.0
  - Entorno de edición de imágenes.

Librerías de funciones

En el desarrollo del proyecto se emplearán las siguientes librerías de funciones:

- Intel Image Processing Library 2.5 (IPL)
  - Librería de funciones para el procesamiento de imágenes.
- Kanade-Lucas-Tomasi Feature Tracker 1.1.5 (KLT)
  - Librería de funciones para la búsqueda y seguimiento de características en imágenes.
- USGS General Cartographic Transformation Package 2.0 (GCTP)
  - Librería de funciones para la conversión de coordenadas entre sistemas de proyección.

NOTA: Las librerías KLT y GCTP han sido portadas de ANSI C a ANSI C++ por el autor del proyecto.

1.5.2 Recursos hardware

Esta sección incluye los *recursos hardware* que serán utilizados en el desarrollo de la aplicación:

Equipo anfitrión

En el desarrollo del proyecto se empleará el siguiente equipo:

- PC compatible
  - PC compatible con procesador Intel PIII - 600MHz, 128MB de memoria estándar y 16MB de memoria de vídeo.

1.5.3 Datos

Como ya se comentó en la sección 1.4, una de las etapas, la destinada al procesamiento de los datos, incluye una tarea dirigida a seleccionar los datos apropiados para el proceso de detección digital de cambios. Los datos empleados en el desarrollo del proyecto han sido proporcionadas por Euromap GmbH, empresa distribuidora en Europa de los datos generados por los satélites IRS (Indian Remote Sensing).

Satélite IRS-1D

El satélite IRS-1D fue lanzado con éxito, en una órbita polar, el 27 de septiembre de 1997 por una lanzadera PSLV y activado a mediados de octubre de 1997 (ver figura 1.11).

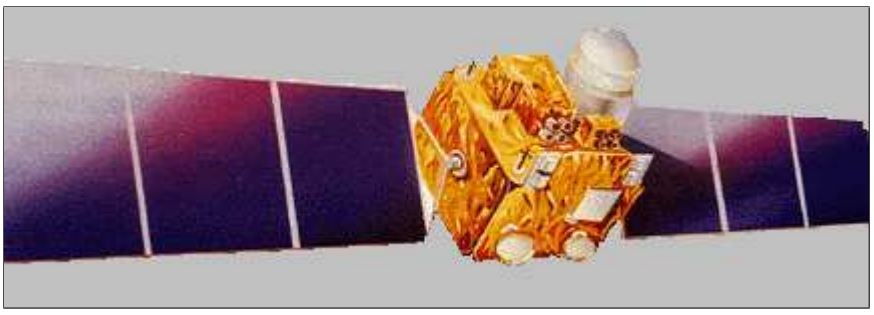


Figura 1.11: Satélite IRS-1D.

Tanto los datos proporcionados por el satélite IRS-1D como las características técnicas del mismo serán detalladas pormenorizadamente en capítulos posteriores.

Parte I

Teledetección

Capítulo 2

Fundamentos

2.1 ¿Qué es la teledetección?

"La teledetección es la ciencia de adquirir información de la superficie terrestre sin estar realmente en contacto con ella. Esto se hace detectando y registrando la energía reflejada o emitida y procesando, analizando y aplicando esa información" [8].

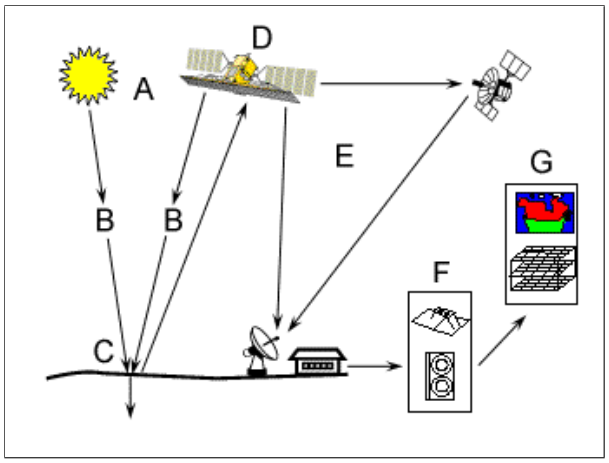


Figura 2.1: Elementos de un sistema de teledetección.

Un sistema de teledetección incluye los siguientes elementos:

1. Fuente de energía/Iluminación (A)

2. Interacciones radiación-atmósfera (B)

3. Interacciones radiación-objeto (C)

4. Registro de la energía (sensor) (D)

5. Transmisión, recepción y procesamiento (E)

6. Interpretación y análisis (F)

7. Aplicación (G)

2.2 Fuente de energía/Iluminación

El primer requisito para la teledetección es tener una fuente que ilumine el objeto (a menos que la energía registrada este siendo emitida por el objeto). La energía, emitida y/o reflejada, está en forma de radiación electromagnética.

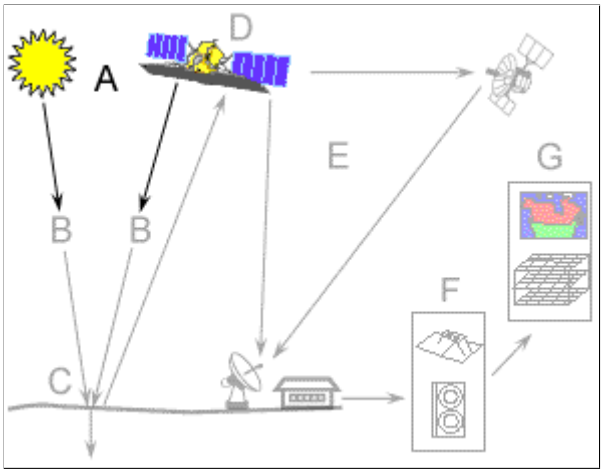


Figura 2.2: Fuente de energía/Iluminación.

Según la teoría ondulatoria, la radiación electromagnética se transmite de un lugar a otro siguiendo un modelo armónico y continuo, a la velocidad de la luz (c) y conteniendo dos campos, el eléctrico (E) y el magnético (M), ortogonales entre sí (ver figura 2.3).

Las características de este flujo energético pueden describirse en términos de dos elementos: la longitud de onda (λ) y la frecuencia (ν). La primera

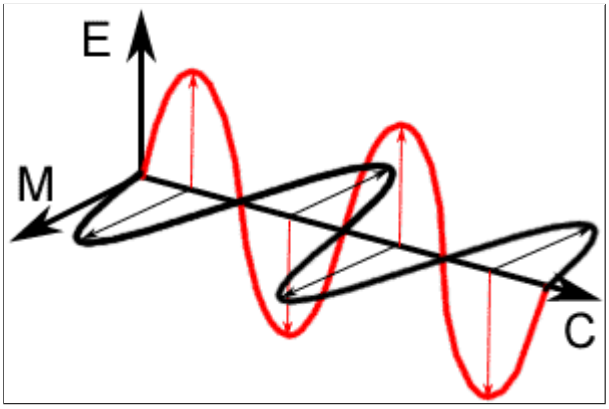


Figura 2.3: Radiación electromagnética.

hace referencia a la distancia entre dos picos sucesivos de una onda, mientras que la segunda designa el número de ciclos que pasan por un punto fijo por unidad de tiempo (ver figura 2.4). La longitud de onda y la frecuencia están relacionados por la siguiente fórmula:

$$c = \lambda\nu \tag{2.1}$$

donde c es la velocidad de la luz (3×10^8 m/s), λ la longitud de onda (m) y ν la frecuencia (Hz).

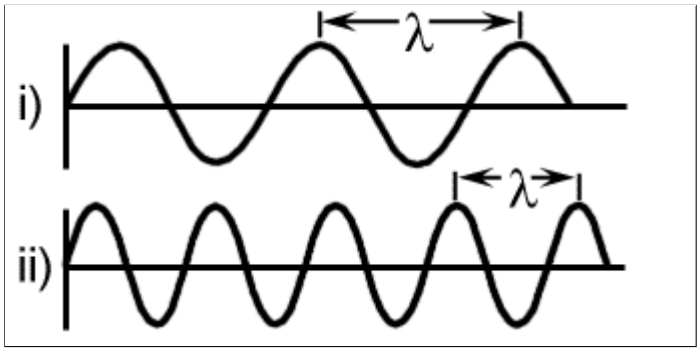


Figura 2.4: Longitud de onda (λ).

Por lo tanto, ambos elementos están inversamente relacionados, de forma que a mayor longitud de onda, menor frecuencia y viceversa. Comprender las características de la radiación electromagnética en términos de su longitud de onda y frecuencia es crucial para entender la información obtenida en el proceso de teledetección.

2.3 El espectro electromagnético

Aunque la sucesión de valores de longitud de onda es continua, suelen establecerse una serie de bandas en donde la radiación electromagnética manifiesta un comportamiento similar (ver figura 2.5). La organización de estas bandas de longitudes de onda o frecuencias se denomina espectro electromagnético, y abarca desde las longitudes de onda más cortas (radiaciones γ y rayos x) hasta las longitudes de onda más largas (microondas y las ondas de radiodifusión).

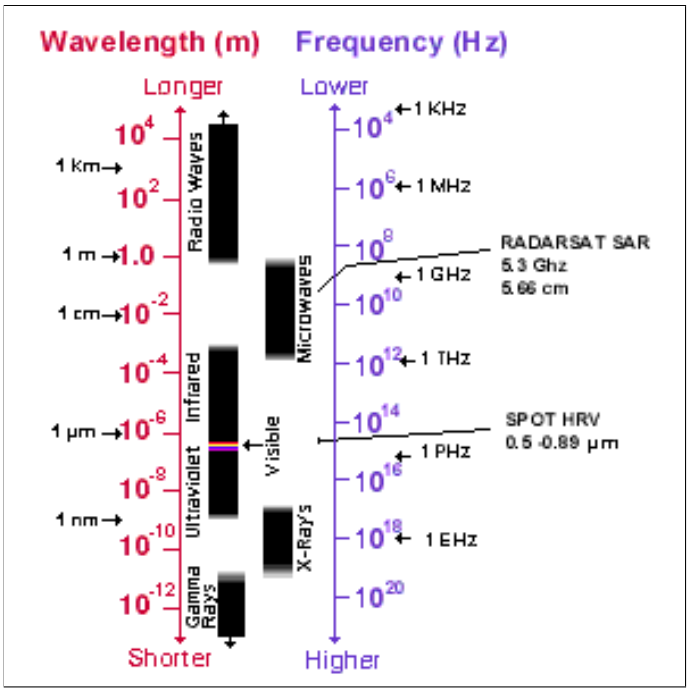


Figura 2.5: Espectro electromagnético.

Hay varias regiones del espectro electromagnético que son útiles para el proceso de *teledetección*:

2.3.1 Región ultravioleta

La región ultravioleta (UV) del espectro cuenta con las longitudes de onda más cortas útiles para el proceso de teledetección. Esta radiación está por debajo de la región violeta (V) del espectro visible, de ahí su nombre. Algunos

materiales de la superficie terrestre, principalmente rocas y minerales, emiten luz visible cuando son iluminados con radiaciones UV.

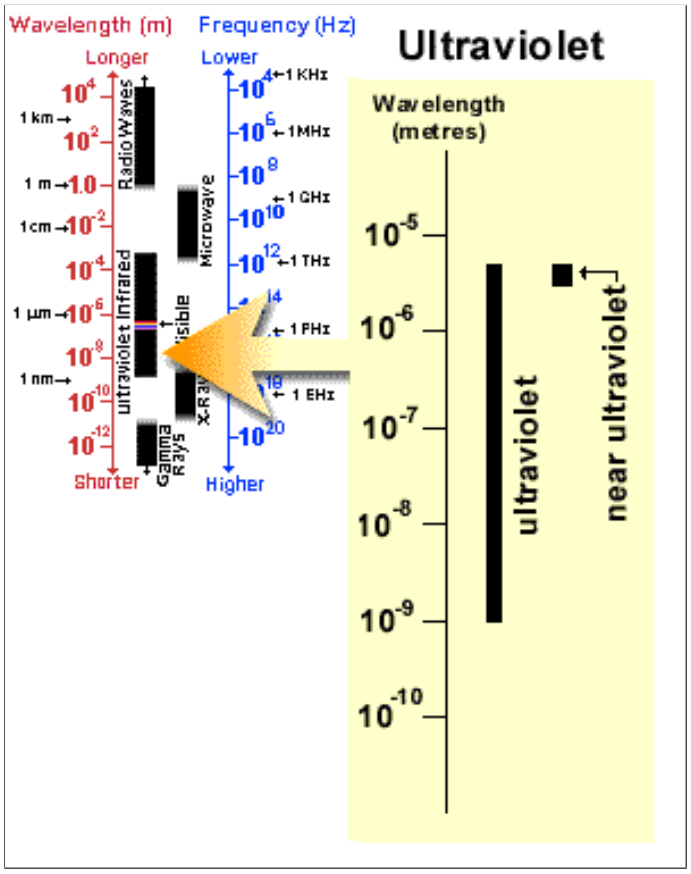


Figura 2.6: Radiaciones UV.

2.3.2 Región visible

La luz que nuestros ojos –nuestros "sensores remotos"– puede detectar pertenece a la región visible del espectro (ver figura 2.7). El tamaño de esta región es pequeño en comparación con el resto del espectro electromagnético. Las longitudes de onda visibles cubren un rango que va desde los 0.4 μm hasta 0.7 μm aproximadamente.

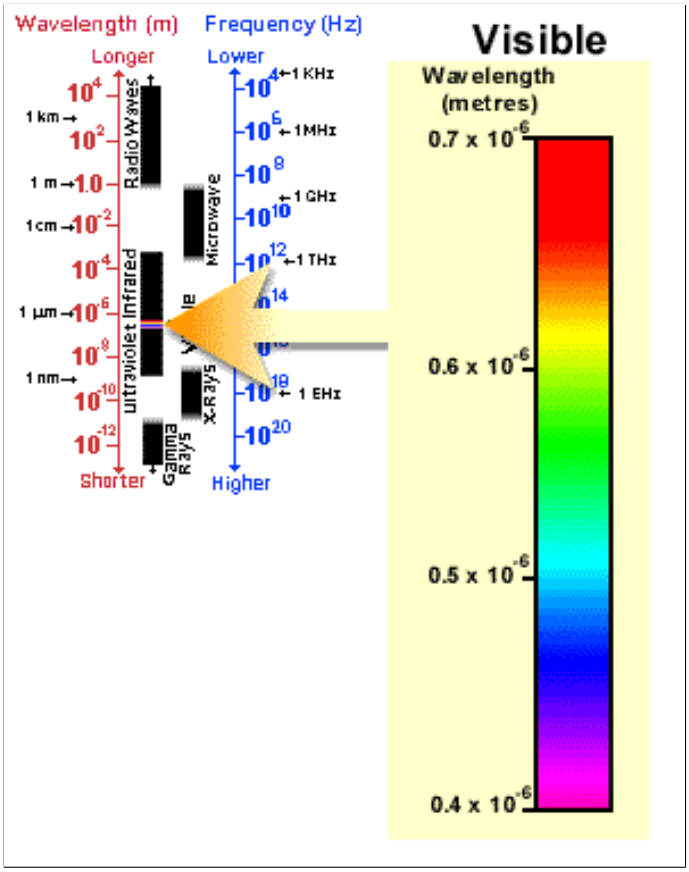


Figura 2.7: Espectro visible.

La longitud de onda más larga del espectro visible se corresponde con el rojo y la más corta con el violeta. Algunas de las longitudes de onda de la región visible del espectro que nosotros percibimos como "colores" son las siguientes:

- `Violeta:` 0.4-0.446 μm
- `Azul (P):` 0.446-0.500 μm
- `Verde (P):` 0.500-0.578 μm
- `Amarillo:` 0.578-0.592 μm
- `Naranja:` 0.592-0.620 μm
- `Rojo (P):` 0.620-0.7 μm

El **azul**, **verde** y **rojo** son los **colores primarios**, reciben este nombre debido a que ningún color primario puede obtenerse como combinación de otros dos, sin embargo, el resto de los colores de la región visible del espectro se pueden obtener como combinación, en diversas proporciones, de estos tres colores.

2.3.3 Región infrarroja

La siguiente región de interés del espectro es la región infrarroja (IR) que cubre un rango que va desde los 0.7 μm hasta 100 μm aproximadamente (más de cien veces la región visible) (ver figura 2.8). La región IR se puede dividir en varias bandas en función de sus propiedades específicas [4]:

- IR cercano o reflejado (0.7-1.3 μm), que incluye el proceso de reflexión de la luz solar. Resulta de especial importancia por su capacidad para discriminar masas vegetales y concentraciones de humedad.
- IR medio (1.3-8 μm), en donde se entremezclan los procesos de reflexión de la luz solar y de emisión de la superficie terrestre. Resulta idóneo para estimar el contenido de humedad en la vegetación y detectar focos de alta temperatura.
- IR lejano o térmico (8-14 μm), que incluye el proceso de emisión de la superficie terrestre. Resulta de especial importancia para detectar el calor proveniente de la mayor parte de las cubiertas terrestres.

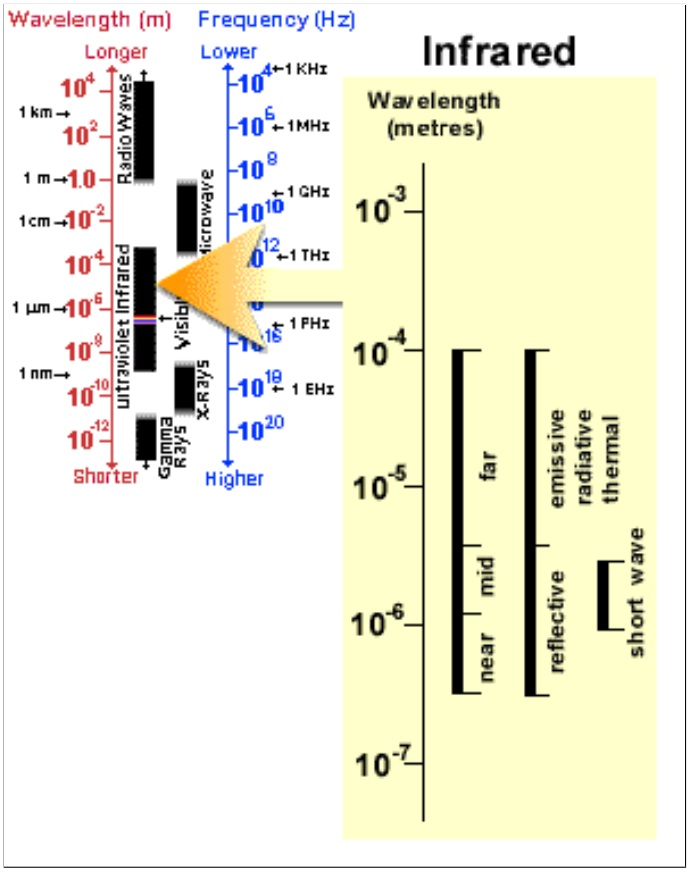


Figura 2.8: Radiaciones IR.

2.4 Interacciones radiación-atmósfera

Antes de que la radiación utilizada durante la teledetección alcance la superficie terrestre, tiene que viajar alguna distancia a través de la atmósfera. Las partículas y gases pueden afectar la radiación y luz incidentes. Estos efectos son causados por los mecanismos de *dispersión* (*scattering*) y *absorción*.

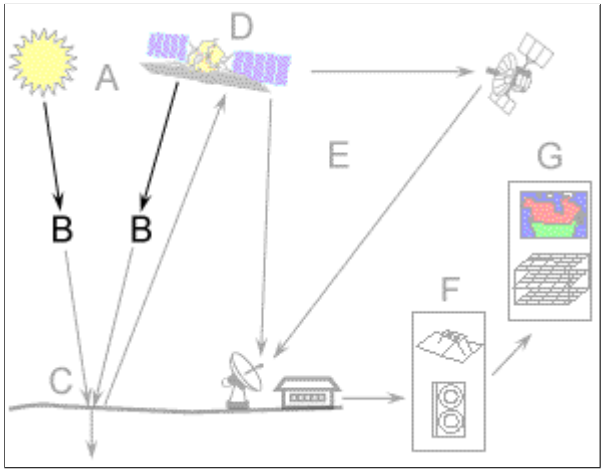


Figura 2.9: Interacciones radiación-atmósfera.

La *dispersión* ocurre cuando partículas o grandes moléculas de gas presentes en la atmósfera interactúan con la radiación electromagnética y provocan la desviación de su trayectoria original (ver figura 2.10). La dispersión depende de varios factores:

- La longitud de onda de la radiación
- La abundancia de partículas o gases
- La distancia que la radiación ha de recorrer a través de la atmósfera (grosor de la atmósfera)

Se pueden dar tres tipos de dispersión:

1. Dispersión *Rayleigh*
2. Dispersión *Mie*
3. Dispersión *no selectiva*

La dispersión *Rayleigh* tiene lugar cuando las partículas son muy pequeñas comparadas con la longitud de onda de la radiación (pequeñas motas de polvo o moléculas de nitrógeno y oxígeno). El efecto *Rayleigh* provoca que la radiaciones electromagnéticas con longitudes de onda más cortas se dispersen mucho más que las de longitudes de onda más largas. La dispersión

Rayleigh es el mecanismo de dispersión predominante en las capas altas de la atmósfera. El hecho de que el cielo sea "azul" durante el día se debe a este fenómeno.

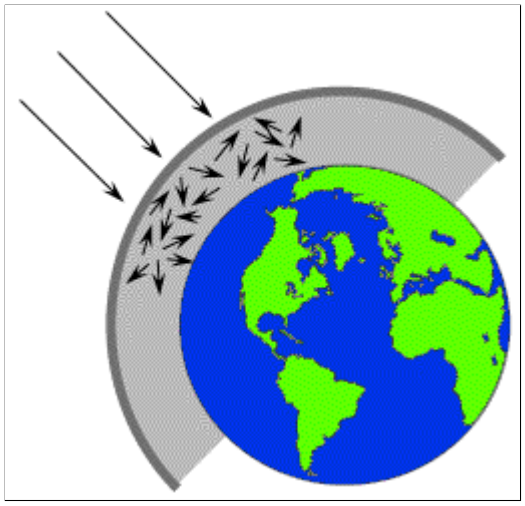


Figura 2.10: Dispersión.

La dispersión *Mie* tiene lugar cuando las partículas son aproximadamente del mismo tamaño que la longitud de onda de la radiación. El polvo, polen, humo y vapor de agua son causas comunes de la dispersión *Mie*, que tiende a afectar a radiaciones con longitudes de onda más largas que las afectadas por la dispersión *Rayleigh*. La dispersión *Mie* tiene lugar generalmente en las capas bajas de la atmósfera donde las partículas son más abundantes, siendo especialmente importante cuando está nublado.

El último mecanismo de dispersión de importancia es el denominado *no selectivo* (ver figura 2.11). Este tiene lugar cuando las partículas son mucho más grandes que la longitud de onda de la radiación. Las gotas de agua y grandes partículas de polvo pueden causar este tipo de dispersión. La dispersión no selectiva toma su nombre del hecho de que todas las radiaciones son dispersadas por igual, independientemente de su longitud de onda. Este tipo de dispersión provoca que la niebla y las nubes aparezcan "blancas" ante nuestros ojos.

Figura 2.11: Dispersión no selectiva.

La *absorción* es el otro mecanismo esencial cuando la radiación electromagnética interactúa con la atmósfera. A diferencia de la dispersión, este fenómeno lo causan las moléculas presentes en la atmósfera al absorber energía en varias longitudes de onda (ver figura 2.12). El ozono, el dióxido de carbono y el vapor de agua son tres de los elementos atmosféricos que absorben radiación.

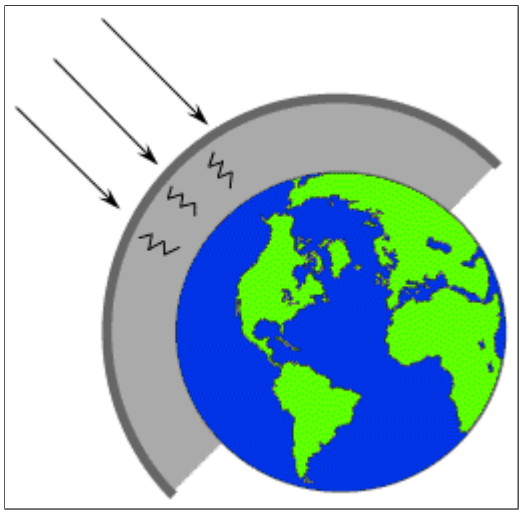


Figura 2.12: Absorción.

El ozono (O_3) sirve para absorber la radiación UV del sol. Sin esta capa protectora en la atmósfera, nuestra piel se quemaría cuando se expusiera a la luz solar. El dióxido de carbono (CO_2) tiende a absorber radiación en el IR lejano (asociado a los procesos de emisión de la superficie terrestre)

lo cual sirve para "atrapar" el calor dentro de la atmósfera[1]. El vapor de agua absorbe gran cantidad de radiación en el IR lejano y microondas. La presencia de vapor de agua en las capas bajas de la atmósfera varía de una localización a otra y de una época del año a otra.

Puesto que estos gases absorben energía electromagnética en regiones muy específicas del espectro, determinan qué regiones del espectro utilizar para aplicaciones de teledetección. Aquellas regiones del espectro que no se ven afectadas gravemente por la absorción atmosférica y, por lo tanto, útiles para sensores remotos, reciben el nombre de *ventanas atmosféricas* (ver figura 2.13).

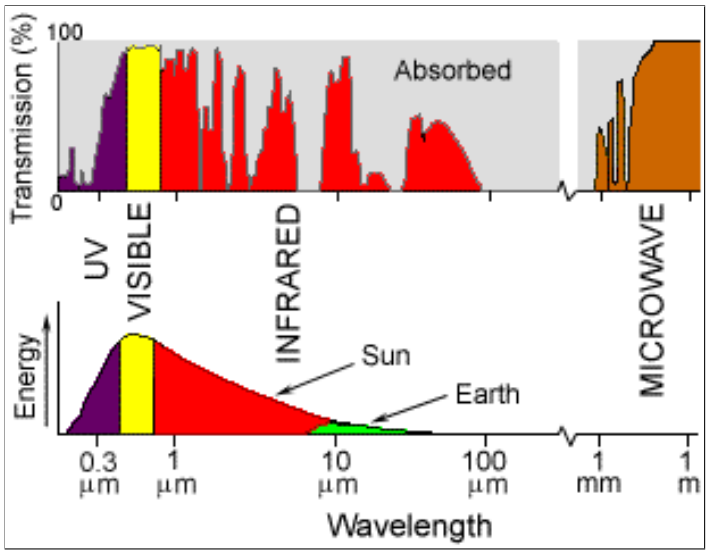


Figura 2.13: Ventanas atmosféricas.

Comprobando las características de las fuentes de energía/radiación más comunes (el sol y la tierra) con las ventanas atmosféricas disponibles podemos definir aquellas longitudes de onda que pueden ser utilizadas de forma efectiva en aplicaciones de teledetección.

[1]Efecto invernadero.

2.5 Interacciones radiación-objeto

La radiación que no es absorbida o dispersada en la atmósfera puede alcanzar e interactuar con la superficie terrestre.

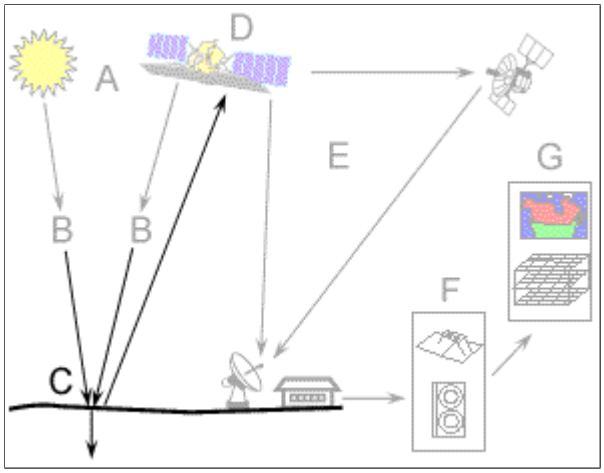


Figura 2.14: Interacciones radiación-objeto.

Se pueden dar tres tipos de interacciones cuando la energía incide sobre la superficie, estas son:

- Absorción (A)
- Transmisión (T)
- Reflexión (R)

La totalidad de la energía incidente interactuará con la superficie en una o más de estas tres formas. La proporción de cada uno dependerá de la longitud de onda de la energía y, del material y estado del objeto (ver figura 2.15).

La *absorción* (A) tiene lugar cuando la radiación es absorbida por el objeto mientras que la *transmisión* (T) tiene lugar cuando la radiación pasa a través del objeto. La *reflexión* (R) se produce cuando la radiación "rebota" en el objeto y es redirigida. En teledetección, la mayoría de las veces estamos interesados en medir la radiación reflejada por los objetos. Hay dos formas en las que un objeto puede reflejar la energía que incide sobre él:

1. Reflexión especular

2. Reflexión difusa

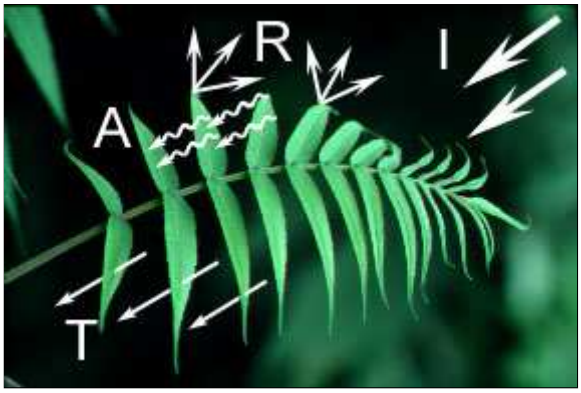


Figura 2.15: Formas de interacción radiación-objetivo.

Cuando la superficie es lisa se produce la reflexión *especular* donde toda (o casi toda) la energía es reflejada en una única dirección (ver figura 2.16).

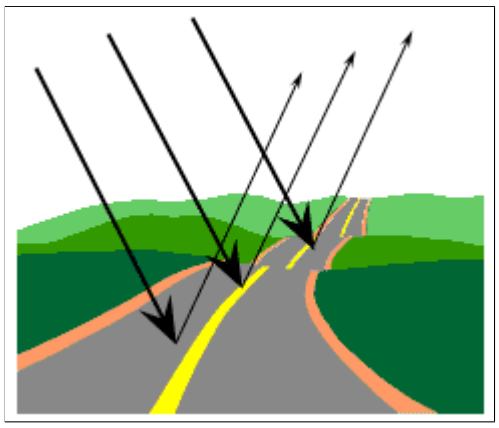


Figura 2.16: Reflexión especular.

La reflexión *difusa* tiene lugar cuando la superficie es rugosa y la energía es reflejada casi uniformemente en todas direcciones (ver figura 2.17).

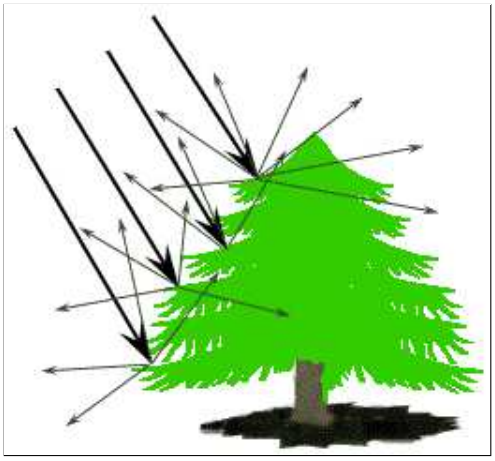


Figura 2.17: Reflexión difusa.

La mayoría de las características de la superficie terrestre están en un punto medio entre los reflectores perfectamente especulares y perfectamente difusos. Si un objeto particular refleja de forma especular o difusa, o de forma intermedia, depende de la rugosidad de su superficie y la longitud de onda de la radiación incidente. Si las longitudes de onda son mucho más pequeñas que las variaciones de la superficie o el tamaño de las partículas que componen la superficie, la reflexión *difusa* predominará.

Dependiendo de la complejidad de la estructura del objeto observado, y las longitudes de onda de las radiaciones implicadas, podemos observar respuestas muy diferentes a los mecanismos de absorción, transmisión y reflexión. Midiendo la energía reflejada (o emitida) por los objetos en la superficie terrestre sobre un conjunto de diferentes longitudes de onda, podemos construir la respuesta espectral para ese objeto (ver figura 2.18).

Comparando los patrones de respuesta de diferentes características, podremos distinguir entre ellas, no podríamos hacer esto, si las comparásemos en una sola longitud de onda. La respuesta espectral puede ser bastante variable, incluso para el mismo tipo de objeto, y puede variar también con el tiempo (por ejemplo, el "verdor" de las hojas) y la localización. Saber donde "mirar" espectralmente y entender los factores que influyen en la respuesta espectral de las características de interés es crítico para interpretar correctamente la interacción de la radiación electromagnética con la superficie.

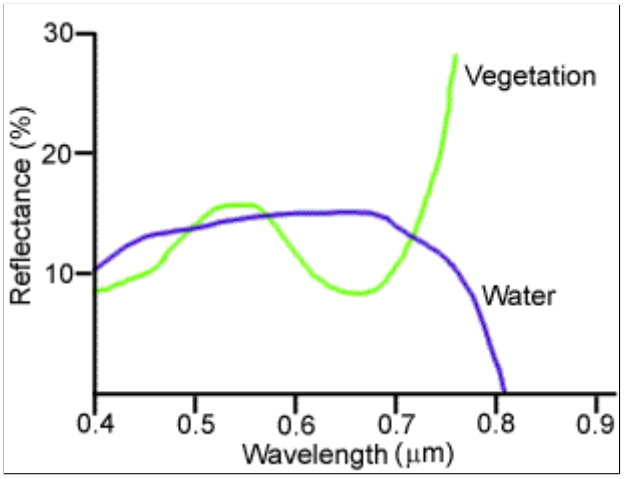


Figura 2.18: Patrones de respuesta espectral (hojas y agua).

2.6 Detección pasiva vs. activa

A lo largo de este capítulo hemos hecho varias referencias al sol como una fuente de energía o radiación. El sol proporciona una fuente de energía muy apropiada para la teledetección. La energía del sol es reflejada, como ocurre para las longitudes de onda visibles, o absorbida y luego emitida, como ocurre para las longitudes de onda del IR lejano (térmico). Los sistemas de teledetección que miden la energía que está disponible de forma natural se denominan sensores *pasivos* (ver figura 2.19). Los sensores *pasivos* sólo pueden utilizarse para detectar energía cuando está disponible de forma natural. Esto sólo puede ocurrir durante el tiempo que el sol esté iluminando la tierra. La energía que es emitida de forma natural (IR térmico) puede ser detectada día y noche, mientras la energía sea lo suficientemente grande para ser registrada.

Por otro lado, los sensores *activos* disponen de su propia fuente de energía (ver figura 2.20). El sensor emite radiaciones que son dirigidas hacia el objeto (región) a ser investigado. La radiación reflejada por el objeto es detectada y medida por el sensor. Una de las ventajas de los sensores activos es la posibilidad de obtener medidas en cualquier momento, independientemente del momento del día o estación. Los sensores activos pueden utilizarse para examinar longitudes de onda que el sol no proporciona suficientemente,

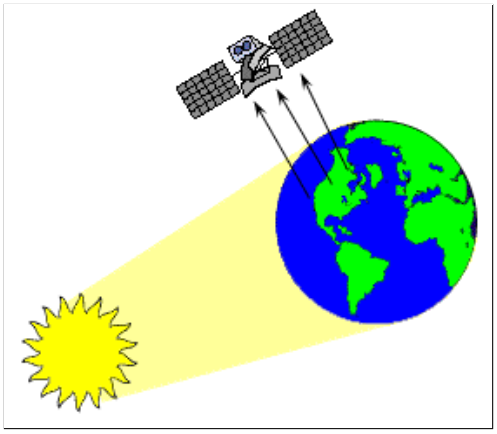


Figura 2.19: Sensores pasivos.

como las microondas, o mejorar la forma en la que el objeto es iluminado. Sin embargo, los sistemas activos requieren la generación de una cantidad bastante importante de energía para iluminar adecuadamente los objetos.

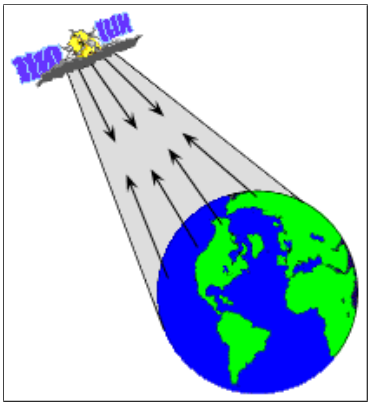


Figura 2.20: Sensores activos.

2.7 Características de las imágenes

Antes de pasar al siguiente capítulo, que trata en mayor detalle los sensores y sus características, necesitamos definir y comprender unos pocos términos y conceptos asociados con las imágenes empleadas en teledetección.

La energía electromagnética se puede detectar fotográficamente o electrónicamente. El proceso fotográfico emplea reacciones químicas sobre la

superficie de la película sensible a la luz para detectar y registrar variaciones de energía. Es importante distinguir entre los términos *imagen* y *fotografía*. Una *imagen* se refiere a cualquier representación pictórica, independientemente de los dispositivos de teledetección que se hayan usado para detectar y registrar la energía electromagnética. Una *fotografía* se refiere especialmente a imágenes que han sido detectadas así como registradas en película fotográfica (ver figura 2.21).

Figura 2.21: Fotografía aérea.

Las "fotos" se registran normalmente sobre un rango de longitudes de onda que va desde los 0.3 μm hasta los 0.9 μm (espectro visible e IR reflejado). Basándonos en esta definición, podemos decir que todas las fotografías son imágenes, pero no todas las imágenes son fotografías. Por lo tanto, a menos que estemos hablando específicamente de una imagen registrada fotográficamente, usaremos el término *imagen*.

Una fotografía podría ser también representada y visualizada en formato digital mediante la subdivisión de la imagen en pequeñas áreas de igual tamaño y forma, denominadas "elementos de pintura" o *píxeles*, y representando el brillo de cada área con un valor numérico o número digital. Los ordenadores visualizan cada valor numérico como diferentes niveles de brillo (ver figura 2.22).

En secciones previas describimos la porción visible del espectro y el concepto de color. Nosotros distinguimos el color debido a que nuestros ojos detectan la totalidad del rango de longitudes de onda visibles y nuestro ce-

rebro procesa la información en colores separados. ¿Qué ocurriría si sólo pudiésemos distinguir un rango de longitudes de onda o colores muy estrecho?. Así es como trabajan muchos sensores. La información de un estrecho rango de longitudes de onda es recogido y almacenado en un *canal* o *banda*. Podemos combinar y visualizar canales de información digital utilizando los tres colores primarios (azul, verde y rojo). Los datos de cada canal representan cada uno de los colores primarios y, dependiendo del brillo relativo (del valor digital) de cada píxel en cada canal, los colores primarios se combinarán en diferentes proporciones para representar distintos colores.

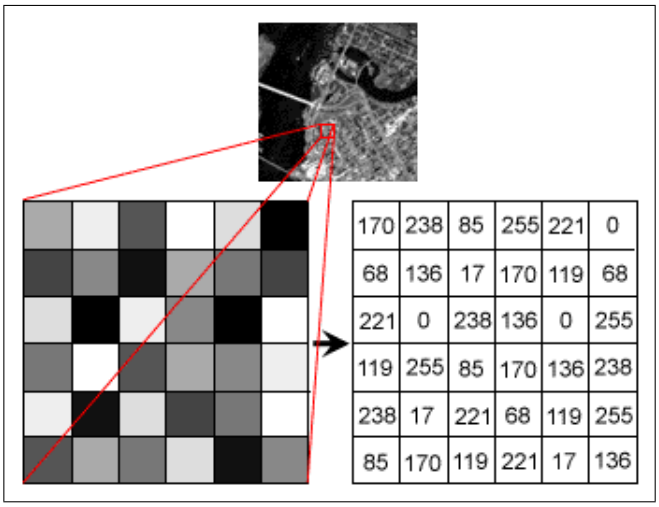


Figura 2.22: Píxel (elemento de pintura).

Cuando utilizamos este método para visualizar un único canal o rango de longitudes de onda, estamos visualizando ese canal realmente a través de los tres colores primarios. Puesto que el nivel de brillo de cada píxel es el mismo para cada color primario, se combinan para formar una imagen en *blanco y negro*, mostrando varias escalas de grises desde el negro hasta el blanco. Cuando visualizamos más de un canal a través de los tres colores primarios, entonces los niveles de brillo pueden ser diferentes para cada combinación de canales/colores primarios y se combinarán para formar una imagen en *color*.

Capítulo 3

Sensores

3.1 Registro de la energía (sensor)

En este capítulo, nos centraremos en este elemento del proceso de teledetección, examinando pormenorizadamente, las características de las plataformas de teledetección, los sensores y los datos que estos proporcionan. Hablaremos brevemente sobre como son procesados los datos una vez que han sido registrados por el sensor.

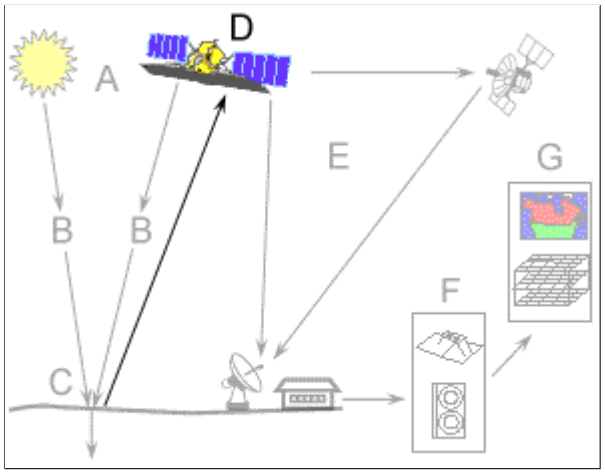


Figura 3.1: Registro de la energía (sensor).

Para que un sensor capte y registre la energía reflejada o emitida por un objeto o superficie, debe residir en una plataforma estable situada sobre el objeto o superficie que está siendo observada. Las plataformas que soportan

los sensores remotos pueden estar situados en un avión, globo sonda (o alguna plataforma dentro de la atmósfera terrestre), o sobre un avión o satélite fuera de la atmósfera terrestre.

Los sensores terrestres se utilizan a menudo, para registrar información detallada sobre la superficie, información que es comparada con la captada por los sensores de un avión o satélite (ver figura 3.2). En algunos casos, esta información puede ser utilizada para caracterizar mejor el objeto que está siendo observado, complementando la información disponible. Los sensores pueden estar colocados sobre una escalera, andamio, edificio, grúa, etc.

Figura 3.2: Sensor terrestre colocado en una grúa.

Las plataformas aéreas que se utilizan habitualmente son los aviones, aunque los helicópteros se utilizan ocasionalmente. Los aviones se utilizan frecuentemente para captar imágenes con un gran nivel de detalle y facilitar la captura de datos, sobre cualquier región de la superficie terrestre y en cualquier momento (ver figura 3.3).

Figura 3.3: Avión.

En el espacio, la teledetección es conducida desde un transbordador espacial (ver figura 3.4) o, más comúnmente, desde satélites (ver figura 3.5). Debido a sus órbitas, los satélites permiten una cobertura periódica y estable de la superficie terrestre.

Figura 3.4: Transbordador espacial.

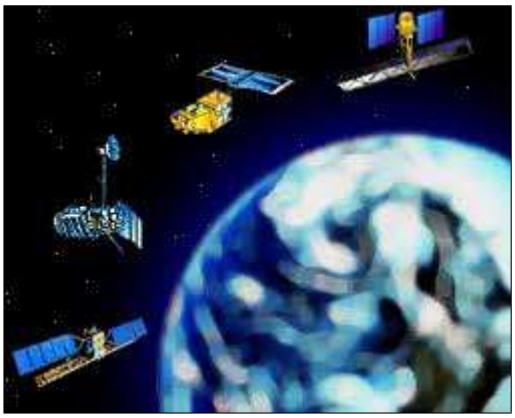


Figura 3.5: Satélites.

El costo es, a menudo, el factor crítico a considerar en la elección del tipo de plataforma.

3.2 Órbitas y coberturas

Como vimos en la sección anterior, los instrumentos de teledetección pueden ser colocados sobre una variedad de plataformas para captar y registrar objetos. Aunque pueden utilizarse plataformas terrestre y aéreas, los satélites proporcionan una gran cantidad de imágenes utilizadas actualmente. Los satélites poseen varias características únicas que los hacen especialmente útiles para la teledetección de la superficie terrestre.

El "camino" seguido por un satélite recibe el nombre de *órbita*. Las órbitas de un satélite se corresponden con la capacidad y objetivos del sensor o sensores que transporta. La selección de la órbita puede variar en términos de altitud y su orientación y rotación relativas a la tierra. Los satélites con altitudes muy elevadas que cubren todo el tiempo la misma región de la superficie terrestre tienen órbitas *geoestacionarias* (ver figura 3.6).

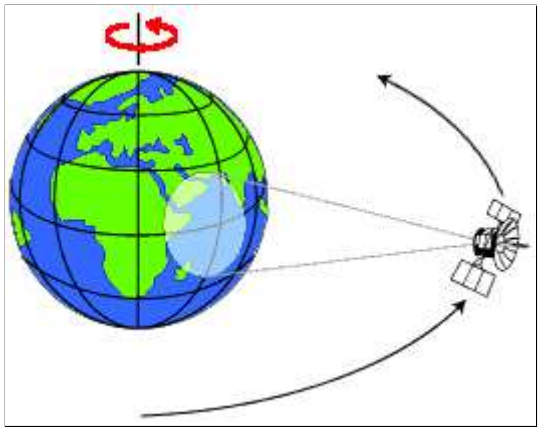


Figura 3.6: Órbitas geoestacionarias.

Los satélites geoestacionarios, en altitudes de aproximadamente 36.000 km, giran a velocidades que se corresponden con la velocidad de rotación de la tierra, por eso parecen inmóviles (estacionarios), con respecto a la superficie terrestre. Esto permite a los satélites observar y captar información continuamente sobre áreas específicas. Los satélites meteorológicos y de telecomunicaciones tienen comúnmente este tipo de órbitas. Debido a su elevada altitud, algunos satélites meteorológicos pueden monitorizar patrones atmosféricos cubriendo completamente un hemisferio de la tierra.

Muchas plataformas de teledetección son diseñadas para seguir una órbita (norte-sur) que, en conjunción con la rotación terrestre (oeste-este), les permite cubrir la mayor parte de la superficie terrestre en un cierto periodo de tiempo. Estas son órbitas *polares*, y reciben su nombre por la ligera inclinación de la órbita respecto a la recta imaginaria que une los polos norte y sur (ver figura 3.7). Muchas de las órbitas de estos satélites están sincronizadas[1] con el sol de forma que cubren cada región de la superficie terrestre

[1]Sun-Synchronous.

en una hora local del día que recibe el nombre de *hora solar local*. Para cualquier latitud, la posición del sol en el cielo, así como la del satélite que pasa por encima, será la misma para la misma estación del año. Esto asegura condiciones de iluminación constantes cuando se adquieren imágenes en una estación específica en años sucesivos, o en un área particular en una serie de días. Esto es un factor importante a la hora de monitorizar cambios entre imágenes o reconstruir mosaicos de imágenes adyacentes, ya que no tienen que ser corregidas para diferentes condiciones de iluminación.

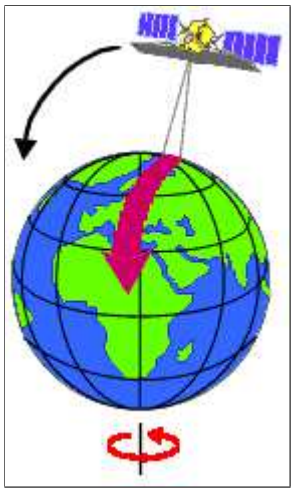


Figura 3.7: Órbitas polares.

La mayoría de los satélites de teledetección actuales están en órbitas polares, lo que significa que el satélite viaja hacia el polo Norte por una lado de la tierra (primera parte de la órbita) y luego hacia el polo Sur por el otro lado (segunda parte de la órbita). Estos desplazamientos reciben el nombre de pasadas *ascendentes* y *descendentes*, respectivamente (ver figura 3.8). Si la órbita esta sincronizada con el sol, la pasada ascendente coincidirá con la cara en penumbra de la tierra (noche) mientras que la pasada descendente lo hará con la cara iluminada de la tierra (día). Los sensores registrarán la energía solar reflejada en la pasada descendente, cuando la iluminación solar está disponible. Los sensores activos que proporcionan su propia iluminación o los sensores pasivos que registran la radiación emitida (térmica) pueden también captar imágenes de la superficie en las pasadas ascendentes.

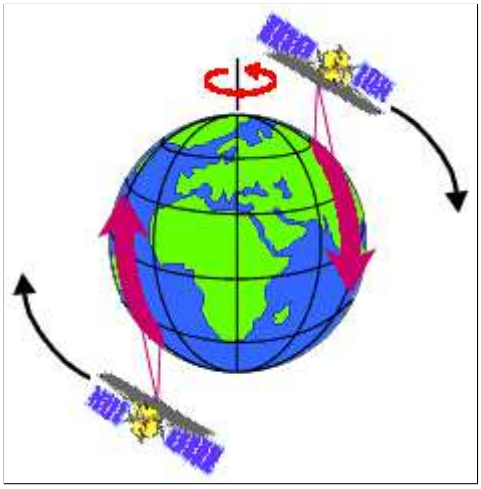


Figura 3.8: Pasadas ascendentes y descendentes.

Como un satélite gira alrededor de la tierra, el sensor "ve" una cierta porción de la superficie terrestre. El área abarcada sobre la superficie recibe el nombre de *cobertura* (ver figura 3.9). La cobertura de los sensores espaciales oscila generalmente entre los diez y cien kilómetros de ancho.

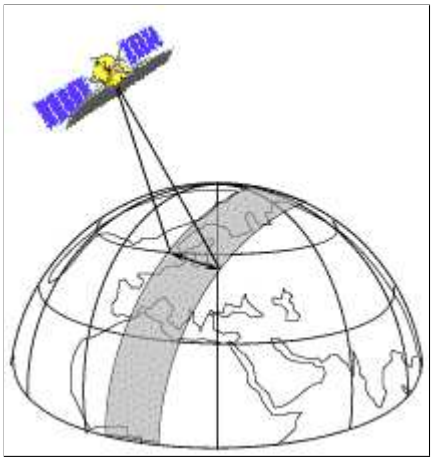


Figura 3.9: Cobertura.

Como el satélite gira alrededor de la tierra de polo a polo, su posición este-oeste no cambiaría si la tierra no rotase. Sin embargo, como vemos desde la tierra, parece que el satélite se desplaza hacia el oeste, esto se debe a que la tierra está rotando (de oeste a este) por debajo de él. Este movi-

miento aparente permite al satélite cubrir una nueva área con cada pasada consecutiva (ver figura 3.10). La órbita del satélite y la rotación de la tierra trabajan juntas para permitir una cobertura total de la superficie terrestre, transcurrido un ciclo completo de órbitas.

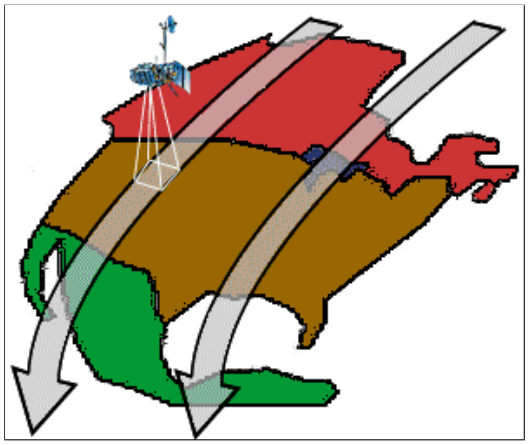


Figura 3.10: Nuevas áreas con cada pasada consecutiva.

Si comenzamos el ciclo de órbitas de un satélite con cualquier pasada seleccionada aleatoriamente, el ciclo de órbitas se completará cuando el satélite repita su "camino", pasando por encima del mismo punto de la superficie terrestre (nadir) en el que comenzó el ciclo. La duración exacta del ciclo de orbitas variará con cada satélite. El intervalo de tiempo requerido para que el satélite complete su ciclo de órbitas no es el mismo que el periodo de "nueva visita". Empleando sensores de cobertura variable, los instrumentos de satélite pueden "ver" un área (fuera de nadir) antes y después de que la órbita pase sobre el objetivo, haciendo así la "nueva visita" en menos tiempo que la duración del ciclo de órbitas. El periodo de "nueva visita" es una consideración importante para un elevado número de aplicaciones de monitorización, especialmente cuando la frecuencia de observación es crítica. En órbitas polares, las regiones próximas a los polos serán visitadas más frecuentemente que la zona ecuatorial debido al solapamiento que tiene lugar en coberturas adyacentes cuando las trayectorias de las órbitas se aproximan (hasta cruzarse) en los polos.

3.3 Resolución espacial, tamaño de pixel

Para algunos instrumentos de teledetección, la distancia entre el objetivo que está siendo observado y la plataforma, juega un papel importante a la hora de determinar el detalle de la información obtenida y el área total observada por el sensor. Los sensores situados en plataformas muy alejadas de sus objetivos, registran generalmente un área mayor, pero no pueden proporcionar un gran detalle. Si comparamos lo que un astronauta ve a bordo de un transbordador con lo que podemos ver desde un avión, el astronauta podría ver de un solo vistazo la totalidad de una provincia o pais, pero no podría distinguir objetos individuales. Volando sobre una ciudad o pueblo, podríamos ver objetos individuales (edificios y coches), pero estaríamos observando un área mucho menor que el astronauta. Hay una diferencia similar entre las imágenes de satélite y las fotografías aéreas.

El detalle discernible en una imagen depende de la *resolución espacial* del sensor y se refiere al tamaño de la característica más pequeña posible que puede ser detectada. La resolución espacial de los sensores pasivos depende principalmente de su *campo de visión instantánea* (IFOV) (ver figura 3.11).

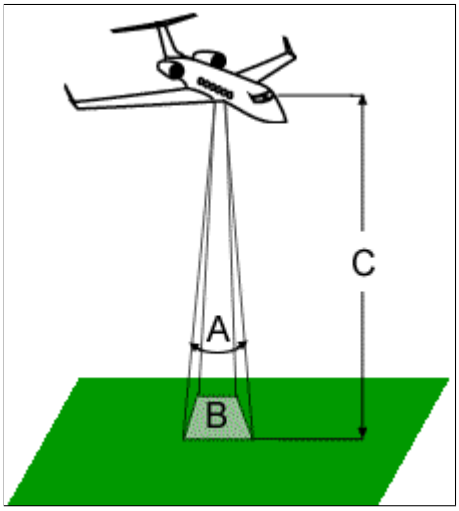


Figura 3.11: Campo de visión instantánea (IFOV).

El IFOV es el ángulo de visión del sensor (A) y determina el área de superficie terrestre que es "vista" desde una altitud determinada en un momento

particular en el tiempo (B). El tamaño del área observada se determina multiplicando el IFOV por la distancia del sensor al suelo (C). Este área sobre el suelo se llama *celda de resolución* y determina la resolución espacial máxima de un sensor. Para detectar una característica homogénea, su tamaño tiene que ser generalmente igual o mayor que la celda de resolución. Sin embargo, características pequeñas, pueden ser detectadas a veces, si su reflectancia domina dentro una celda de resolución (detección subpíxel).

Como mencionamos en el capítulo 2, la mayoría de las imágenes de teledetección están compuestas de una matriz de "elementos de pintura", o píxeles, que son las unidades más pequeñas de una imagen. Los píxeles de la imagen son normalmente cuadrados y representan una cierta área sobre la imagen. Es importante distinguir entre el tamaño de píxel y resolución espacial (no son intercambiables). Si un sensor tiene una resolución espacial de 20 metros y una imagen de ese sensor es visualizada a máxima resolución, cada píxel representa un área de 20×20 metros sobre el suelo. En este caso el tamaño de pixel y la resolución son el mismo. Sin embargo, es posible visualizar una imagen con un tamaño de pixel diferente que la resolución. Muchos mosaicos de imágenes de satélite tienen sus píxeles promediados para representar áreas más grandes, aunque la resolución espacial original del sensor que recogía las imágenes fuese el mismo.

3.4 Resolución espectral

En el capítulo 2 hablamos sobre la respuesta espectral y las curvas de emisividad espectral que caracterizan la reflectancia y/o emitancia de una característica u objeto sobre una variedad de longitudes de onda. Diferentes clases de características y detalles de una imagen pueden ser a menudo distinguidas comparando sus repuestas sobre distintos rangos de longitudes de onda. Gran cantidad de clases, como el agua y la vegetación, pueden separarse usualmente utilizando rangos muy amplios de longitudes de onda –espectro visible e IR cercano– como vimos en la sección 2.5. Otras clases más específicas, como diferentes tipos de rocas, no pueden distinguirse fácilmente utilizando cualquiera de estos rangos tan amplios de longitudes de onda y requerirían la comparación en rangos de longitudes de onda mucho más estrictos para

separarlas (ver figura 3.12). Así, necesitaríamos un sensor con una *resolución espectral* muy alta. La resolución espectral describe la capacidad del sensor para definir intervalos de longitudes de onda adecuados. La adecuación de la resolución espectral se corresponde con el ajuste del rango de longitudes de onda para un canal o banda particular.

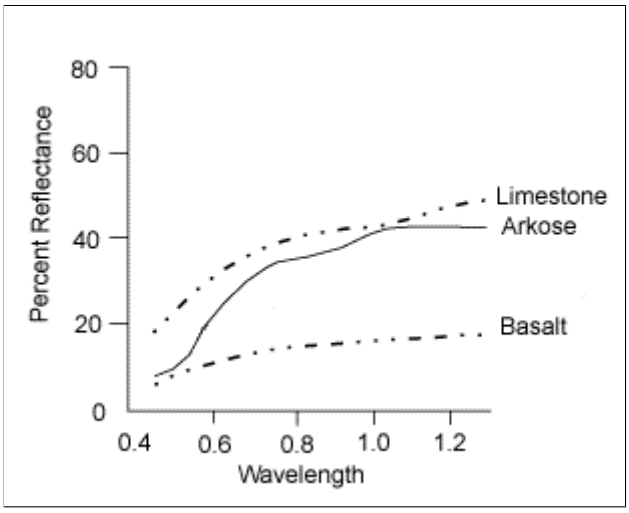


Figura 3.12: Respuesta espectral (diferentes tipos de rocas).

La película en *blanco y negro* registra longitudes de onda de toda o casi toda la región visible del espectro electromagnético. Su resolución espectral es bastante tosca, ya que varias longitudes de onda del espectro visible no se distinguen individualmente, registrándose la totalidad de la reflectancia captada dentro de la porción visible. La película en *color* es también sensible a la energía reflejada sobre la porción visible del espectro, pero tiene una resolución espectral más alta, ya que es sensible de forma individual a la energía reflejada en las longitudes de onda del espectro visible correspondientes al azul, verde y rojo (ver figura 3.13). Así, puede representar características basadas en su reflectancia en cada uno de estos rangos de longitudes de onda (basada en su color).

Muchos sistemas de teledetección registran energía de varios rangos separados de longitudes de onda en varias resoluciones espectrales. Estos sistemas reciben el nombre de sensores *multi-espectrales* y serán descritos con detalle en secciones posteriores. Sensores multi-espectrales avanzados, denominados

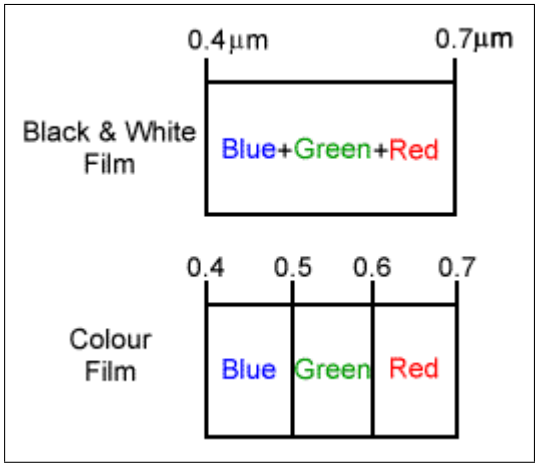


Figura 3.13: Resolución espectral.

sensores *hiper-espectrales*, detectan cientos de bandas espectrales muy estrechas del espectro electromagnético visible, IR cercano e IR medio. Su elevada resolución espectral facilita la discriminación precisa de diferentes objetivos basada en su respuesta espectral para cada una de estas bandas.

3.5 Resolución radiométrica

Mientras que la distribución de los píxeles describen la estructura espacial de una imagen, las características radiométricas describen el contenido de información real en una imagen. Cada vez que se adquiere una imagen sobre una película o por un sensor, su sensibilidad a la magnitud de la energía electromagnética determina la *resolución radiométrica*. La resolución radiométrica de un sensor describe su capacidad para discriminar diferencias muy leves en la energía. Cuanto mejor sea la resolución radiométrica de un sensor, más sensible es para detectar pequeñas diferencias en la energía reflejada o emitida.

Las imágenes son representadas mediante números digitales positivos que van desde el 0 hasta una determinada potencia de 2 (menos uno). Este rango se corresponde con el número de bits empleados para codificar los números en formato binario. El número máximo de niveles de brillo disponibles depende del número de bits utilizados en la representación de la energía registrada.

Así, si un sensor utiliza 8 bits para registrar los datos, habría $2^8 = 256$ valores digitales disponibles (de 0 a 255 niveles de brillo). Sin embargo, si utiliza 4 bits, el número de valores digitales disponibles es $2^4 = 16$ (de 0 a 15 niveles de brillo). Así, la resolución radiométrica sería mucho menor. Las imágenes son visualizadas generalmente en un rango de niveles de gris, con el 0 para representar el negro y el valor máximo (por ejemplo, 255 para una resolución de 8 bits) para representar el blanco. Comparando dos imágenes con una resolución de 2 y 8 bits respectivamente (ver figuras 3.14 y 3.15), podemos observar que hay una diferencia importante en el nivel de detalle discernible dependiendo de sus resoluciones radiométricas.

Figura 3.14: 2 bits de resolución.

Figura 3.15: 8 bits de resolución.

3.6 Resolución temporal

Además de la resolución espacial, espectral y radiométrica, la *resolución temporal* es otro concepto importante a considerar en un sistema de teledetección. Aludimos a esta idea en la sección 3.2 cuando hablamos del concepto de periodo de "nueva visita", que se refiere al tiempo que emplea un satélite para completar un ciclo de órbitas. El periodo de "nueva visita" de un satélite es, por lo general, de varios días. Por lo tanto, la resolución temporal absoluta de un sistema de teledetección coincide con este periodo. Sin embargo, debido

al grado de solape en la cobertura de órbitas adyacentes para la mayor parte de los satélites y el incremento de este solape cuando aumenta la latitud, algunas áreas de la Tierra tienden a ser "visitadas" con una mayor frecuencia. También, algunos sistemas de satélite son capaces de orientar sus sensores para observar el mismo área en órbitas diferentes del satélite, separadas por periodos de uno a cinco días. Así, la resolución temporal real de un sensor depende de una variedad de factores, incluyendo las características del satélite/sensor, el solape en la cobertura de órbitas adyacentes y la latitud (ver figura 3.16).

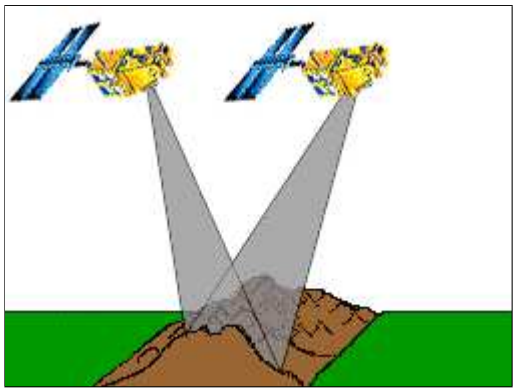


Figura 3.16: Resolución temporal.

La posibilidad de captar imágenes del mismo área de la superficie terrestre en diferentes periodos de tiempo constituye uno de los factores más importantes a la hora de aplicar los datos obtenidos en teledetección. Las características espectrales de los objetos pueden cambiar con el tiempo, y estos cambios pueden ser detectados mediante la captura y comparación multitemporal de imágenes. Por ejemplo, durante la estación de cultivo, muchas especies agrícolas están en continuo estado de cambio y nuestra capacidad para monitorizar estas sutiles alteraciones utilizando teledetección depende de *cómo* y *cuando* captamos las imágenes. La observación continuada nos permite monitorizar los cambios que tienen lugar en la superficie terrestre, tanto si ocurren de forma natural (por ejemplo, incendios, etc.) o inducidos por humanos (por ejemplo, cambios urbanos, etc.). Por lo tanto, el factor tiempo en observación es crítico y depende, fundamentalmente, del tipo de aplicación.

3.7 Cámaras y fotografía aérea

Las cámaras y su uso para la fotografía aérea son el más simple y antiguo de los sensores utilizados para la teledetección de la superficie terrestre. Las cámaras son sistemas de enfoque que adquieren una instantánea (*snapshot*) de un área (A), de la superficie (ver figura 3.17). Los sistemas de cámara son sensores ópticos pasivos que utilizan una lente (B) (o sistema de lentes denominado óptica) para formar una imagen en el plano focal (C).

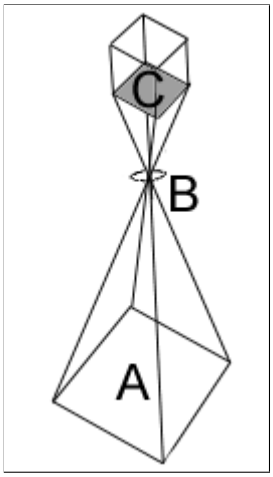


Figura 3.17: Sistema de enfoque.

Las películas *fotográficas* son sensibles a la luz con longitudes de onda que van desde los 0.3 μm hasta los 0.9 μm abarcando las radiaciones UV, visibles y del IR cercano. Las películas *pancromáticas* producen imágenes en blanco y negro y constituye el tipo de película más utilizada en fotografía aérea. La fotografía UV también utiliza película pancromática, pero se utiliza un filtro fotográfico para absorber y bloquear la energía visible que alcanza la película. Como consecuencia, sólo la reflectancia UV de los objetivos se registra. La fotografía UV no se utiliza habitualmente, debido a la dispersión y absorción atmosférica que tiene lugar en esta región del espectro. La fotografía IR en blanco y negro utiliza películas sensibles al rango de longitudes de onda que va desde los 0.3 μm hasta los 0.9 μm y es útil para detectar diferencias en las cubiertas de vegetación, debido a su sensibilidad a la reflectancia IR.

Las fotografías en color o falso color (IR color) implican el uso de tres

capas de película, cada una de ellas sensibles a diferentes rangos de luz. Para una fotografía en color normal, las capas son sensibles a la luz azul, verde y roja –lo mismo que nuestros ojos. Estas fotos aparecen de la misma forma que nuestros ojos ven el entorno que nos rodea (ver figura 3.18). En la fotografía IR color, las tres capas de emulsión son sensibles al verde, rojo, y la porción fotográfica del IR cercano, que son procesados para aparecer como azul, verde y rojo, respectivamente. En una fotografía en falso color, los objetivos con alta reflectancia IR aparecen en rojo, aquellos con alta reflectancia roja aparecen en verde, y aquellos con una alta reflectancia verde aparecen en azul, dándonos una representación "falsa" de los objetivos relativa al color con los que normalmente los percibimos (ver figura 3.19).

Figura 3.18: Fotografía en color.

Figura 3.19: Fotografía en falso color.

Las cámaras se utilizan sobre una variedad de plataformas incluyendo soportes terrestres, aviones y plataformas espaciales. Las fotografías con alto nivel de detalle tomadas desde un avión son útiles para muchas aplicaciones donde se requiere la identificación de detalles u objetivos pequeños. La cobertura terrestre de una foto depende de varios factores, incluyendo la distancia focal de la lente, la altitud de la plataforma y el formato y tamaño de la película. La distancia focal controla el campo de visión angular de la lente (similar al concepto de campo de visión instantánea discutido en la sección 3.3) y determina el área "vista" por la cámara. Las longitudes focales típicas utilizadas son 90 mm, 210 mm y más habitualmente, 152 mm. Cuanto

mayor sea la distancia focal, más pequeña será el área cubierta sobre el suelo, pero con mayor detalle. El área cubierta también depende de la altitud de la plataforma. Para altitudes elevadas, una cámara "verá" un área más grande sobre el suelo que a altitudes más bajas, pero con un nivel de detalle menor. Las fotos aéreas pueden proporcionar detalles con resoluciones espaciales inferiores a los 50 cm. La resolución espacial exacta de una foto varía como una función compleja de muchos factores que varían a su vez con cada adquisición.

La mayoría de las fotografías aéreas se clasifican en *oblicuas* o *verticales*, dependiendo de la orientación relativa entre el suelo y la cámara en el momento de la adquisición. Las fotografías aéreas *oblicuas* se toman con la cámara colocada sobre el costado del avión. Las fotografías muy oblicuas incluyen usualmente el horizonte mientras que las fotografías poco oblicuas no lo hacen. Las fotografías oblicuas pueden resultar útiles para cubrir áreas muy amplias en una única imagen, y para delinear el relieve y escala del terreno. Sin embargo, no se utilizan habitualmente ya que las distorsiones (perspectiva) imposibilitan la medida de distancias, áreas y elevaciones (ver figura 3.20).

Figura 3.20: Fotografía aérea oblicua.

Las fotografías *verticales* tomadas con una cámara de una única lente constituyen el uso más común de la fotografía aérea para propósitos de teledetección y cartografía. Estas cámaras son construidas especialmente para capturar una rápida secuencia de fotografías, limitando la distorsión geométrica. Estas cámaras están unidas a menudo a sistemas de navegación a

bordo de plataformas aéreas, para permitir que las coordenadas geográficas se asignen instantáneamente a cada fotografía. La mayoría de estos sistemas también incluyen mecanismos que compensan los efectos del movimiento del avión respecto al suelo, para limitar la distorsión tanto como sea posible.

Cuando se obtiene una fotografía aérea vertical, el avión vuela normalmente en una serie de líneas, denominadas líneas de vuelo (ver figura 3.21). Las fotos de la superficie se toman en una rápida sucesión, a menudo con un 50-60% de solape (A) entre fotos sucesivas. El solape asegura la cobertura total a lo largo de la línea de vuelo y también facilita la *visión estereoscópica*. Pares sucesivos de fotos muestran la región de solape desde diferentes perspectivas y puede ser vista a través de un dispositivo denominado estereoscópio que permite obtener una visión tridimensional del área, llamada *modelo estéreo*. Muchas aplicaciones de la fotografía aérea utilizan cobertura estereoscópica y visión estéreo.

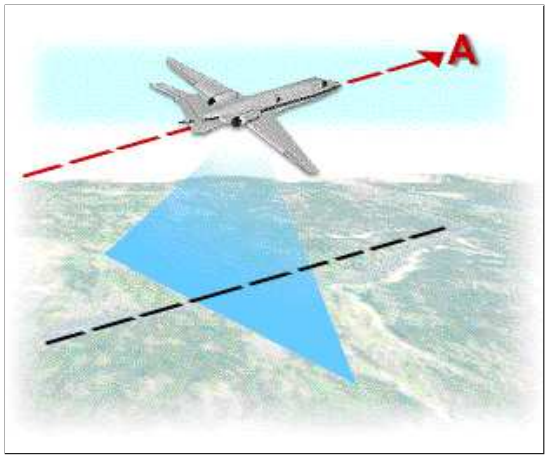


Figura 3.21: Línea de vuelo.

Las fotografías aéreas son útiles cuando el detalle espacial es más crítico que la información espectral, ya que su resolución espectral es generalmente tosca comparada con la de los datos capturados con sensores electrónicos. La geometría de las fotografías verticales se entiende adecuadamente y es posible hacer medidas muy exactas sobre ellas, para una amplia variedad de aplicaciones (geología, cartografía, etc.). La ciencia de hacer medidas sobre fotografías se denomina *fotogrametría* y se ha venido realizando extensiva-

mente desde los principios de la fotografía aérea. Las fotos son interpretadas manualmente por un analista humano. Pueden ser escaneadas para crear una imagen digital y entonces analizada en un sistema informático.

La fotografía *multi-banda* utiliza múltiples sistemas de lentes con diferentes combinaciones de películas/filtros para adquirir fotos simultáneamente en un conjunto de distintos rangos espectrales. La ventaja de este tipo de cámaras es la capacidad para registrar energía reflejada separadamente, en rangos discretos de longitudes de onda. Así, proporciona potencialmente una mejor separación e identificación de varias características. Sin embargo, el análisis simultáneo de estas múltiples fotografías puede resultar problemático.

Las *cámaras digitales*, que registran radiación electromagnética electrónicamente, difieren sensiblemente de sus equivalentes que utilizan película. En lugar de usar película, las cámaras digitales utilizan una malla de CCDs (dispositivo de acoplamiento de cargas) que responden individualmente a la radiación electromagnética. La energía que alcanza la superficie de los CCDs provoca la generación de una carga electrónica proporcional en magnitud al "brillo" del área observada. Se asigna un número digital a cada pixel (para cada banda espectral), basado en la magnitud de la carga electrónica. El formato digital de la imagen de salida es sensible al análisis digital y al almacenamiento en un entorno informático. Las cámaras digitales también proporcionan una mayor velocidad (turn-around) de adquisición y recuperación de los datos, y permite un mayor control de la resolución espectral. Aunque los parámetros varían, los sistemas digitales son capaces de captar datos con una resolución espacial de 30 cm, y con una resolución espectral que va desde los 0.012 μm hasta los 0.3 μm. El tamaño del array de píxeles varía de unos sistemas a otros, pero oscila típicamente entre los 512×512 y 2048×2048 píxeles.

3.8 Exploración multi-espectral

Muchos sensores de teledetección electrónicos adquieren datos utilizando *sistemas de exploración*[2], que emplean un sensor con un estrecho campo de

[2]Escáners.

visión instantánea (IFOV) que barre el terreno para producir imágenes de la superficie. Los sistemas de exploración pueden utilizarse tanto en plataformas aéreas como de satélite y tienen esencialmente los mismos principios de operación. Un sistema de exploración utilizado para captar datos sobre diferentes rangos de longitudes de onda recibe el nombre de *escáner multi-espectral* (MSS), constituyendo uno de los sistemas de exploración más utilizados. Hay dos modos o métodos de exploración importantes empleados para adquirir imágenes multi-espectrales –la exploración de *barrido* y la exploración de *empuje*.

Los sistemas de exploración de *barrido* exploran la Tierra en una serie de líneas perpendiculares a la dirección en la que se mueve la plataforma del sensor (ver figura 3.22). Cada línea se explora de un lado a otro del sensor, utilizando un espejo rotatorio (A). Como la plataforma se mueve sobre la superficie, exploraciones sucesivas generan una imagen bidimensional de la superficie terrestre. La radiación reflejada o emitida se separa en sus componentes espectrales que se detectan independientemente. La radiación UV, visible, IR cercano e IR térmico se disgregan en sus longitudes de onda constituyentes. Un banco de detectores internos (B), cada uno sensible a un rango específico de longitudes de onda, detectan y miden la energía para cada banda espectral y entonces, como una señal eléctrica, se convierte a datos digitales y se registra para el subsiguiente procesamiento informático.

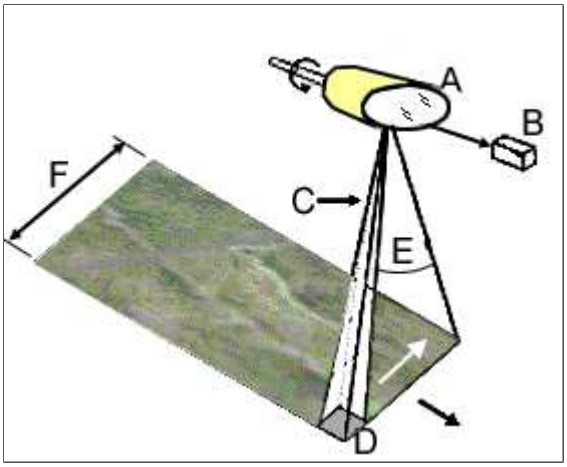


Figura 3.22: Exploración de barrido.

El IFOV (C) del sensor y la altitud de la plataforma determinan la celda de resolución terrestre observada (D), y por lo tanto la resolución espacial. El campo de visión angular (E) se corresponde con el barrido del espejo, medido en grados, utilizado para registrar una línea explorada, y determina el ancho de la superficie observada. Los escáners aerotransportados barren típicamente ángulos bastante amplios (entre 90 y 120 grados), mientras que a los escáners situados en satélites, debido a su elevada altitud, les sobra con un ángulo relativamente pequeño (entre 10 y 20 grados) para cubrir una amplia región. Como la distancia del sensor al objetivo aumenta en los bordes de la superficie observada, las celdas de resolución terrestre también se agrandan e introducen *distorsiones geométricas* en la imagen. También, el tiempo del que dispone el sensor para explorar una celda de resolución terrestre (llamado tiempo de morada[3]) es generalmente bastante corto, e influye en el diseño de la resolución espacial, espectral y radiométrica del sensor.

Los sistemas de exploración de *empuje* utilizan también el movimiento de la plataforma para registrar líneas exploradas sucesivamente y generar una imagen bidimensional, perpendicular a la dirección del vuelo. Sin embargo, en lugar de un espejo de exploración, utilizan un array de detectores (A) localizado en el plano focal de la imagen (B) que son "empujados" en la dirección del vuelo. Estos sistemas se denominan también *escáners de escoba*, ya que el movimiento del array de detectores es análogo al de las cerdas de una escoba que está siendo empujada por el suelo (ver figura 3.23). Cada detector individual mide la energía para una única celda de resolución terrestre (D), así el tamaño e IFOV de los detectores determinan la resolución espacial del sistema. Se necesita un array para medir cada banda espectral o canal. Para cada línea explorada, la energía detectada por cada detector (de cada array) es muestreada electrónicamente y registrada digitalmente.

[3] dwell time.

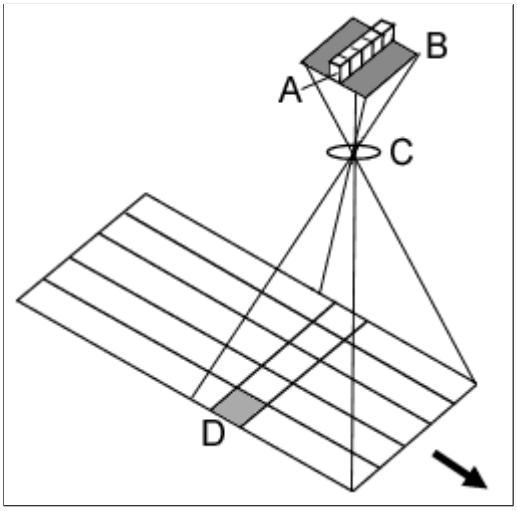


Figura 3.23: Exploración de empuje.

Los escáners de empuje con arrays lineales tienen varias ventajas sobre los escáners de barrido. El array de detectores combinado con el movimiento de barrido permite a cada detector "ver" y medir la energía de cada celda de resolución terrestre durante un periodo más largo de tiempo (tiempo de morada). Esto permite que se detecte más energía y mejora la resolución radiométrica. El incremento del tiempo de morada facilita también IFOVs más pequeños y anchos de banda más estrechos para cada detector. Así, se puede lograr una mejor resolución espacial y espectral sin afectar a la resolución radiométrica. Puesto que los detectores son normalmente dispositivos micro-electrónicos de estado sólido, son generalmente más pequeños, ligeros, requieren menos energía, y son más fiables y duraderos ya que no tienen partes móviles. Por otro lado, la calibración de miles de detectores para lograr una sensibilidad uniforme es necesaria y complicada.

Independientemente de si el sistema de exploración utilizado es cualquiera de estos dos tipos, ambos tienen varias ventajas sobre los sistemas fotográficos. El rango espectral de los sistemas fotográficos esta restringido a las regiones visible e IR cercano, mientras que los sistemas MSS pueden extender este rango al IR térmico. También son capaces de obtener una resolución espectral mucho más alta que los sistemas fotográficos. Los sistemas fotográficos multi-banda o multi-espectrales utilizan sistemas de lentes (óp-

ticas) independientes para adquirir cada banda espectral. Esto puede causar problemas al asegurar que diferentes bandas son comparables tanto espacialmente como radiométricamente, y con el registro de múltiples imágenes. Los sistemas MSS adquieren todas las bandas espectrales simultáneamente a través del mismo sistema óptico para minimizar estos problemas. Los sistemas fotográficos registran la energía detectada mediante un proceso fotoquímico difícil de medir y poco consistente. Puesto que los datos MSS son registrados electrónicamente, es más fácil determinar la cantidad específica de energía medida, y pueden registrar un mayor rango de valores en un formato digital. Los sistemas fotográficos requieren un suministro continuo de película y procesamiento en tierra después de que las fotos hayan sido tomadas. Por contra, el registro digital en los sistemas MSS facilita la transmisión de datos a las estaciones de recepción y el procesamiento inmediato de los datos en un entorno informático.

3.9 Visión térmica

Muchos sistemas multi-espectrales (MMS) detectan radiación en el IR térmico además de las porciones visible y IR reflejado del espectro. Sin embargo, la teledetección de energía emitida desde la superficie terrestre en el IR térmico (desde los 3 μm hasta los 15 μm) es diferente que la detección de la energía reflejada. Los sensores *térmicos* utilizan foto detectores sensibles al contacto directo de los fotones, para detectar la radiación térmica emitida (ver figura 3.24). Los detectores son enfriados a temperaturas cercanas al cero absoluto para limitar sus propias emisiones térmicas. Los sensores térmicos miden esencialmente la temperatura de la superficie y las propiedades térmicas de los objetivos.

Los sensores térmicos son típicamente escáners de barrido que detectan solamente la radiación emitida en la porción térmica del espectro. Los sensores térmicos emplean una o más temperaturas de referencia para comparar la radiación detectada, por eso pueden relacionarse con la temperatura radiante absoluta. Los datos son registrados generalmente sobre película y/o cinta magnética y la resolución térmica de los sensores actuales puede alcanzar los 0.1 grados.

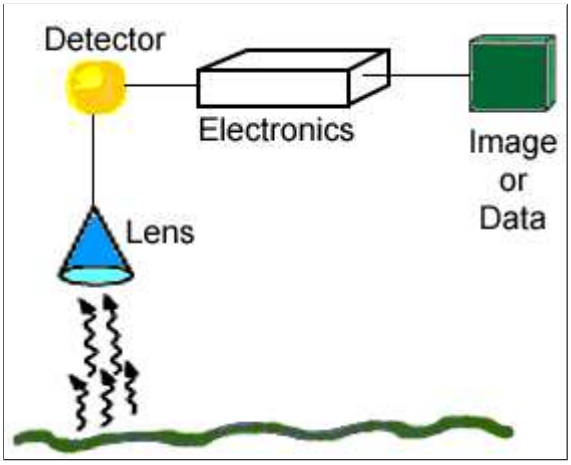


Figura 3.24: Sensores térmicos.

Para el análisis, una imagen de temperaturas relativas (termograma) se representa con niveles de gris, con las temperaturas cálidas mostradas en tonos claros, y las temperaturas frías en tonos oscuros (ver figura 3.25). Las imágenes que representan las diferencias de temperaturas relativas en sus correspondientes localizaciones espaciales son suficientes para la mayoría de las aplicaciones. Las medidas de temperatura absoluta pueden calcularse, pero requieren la calibración y medida exacta de las temperaturas de referencia y conocimiento detallado de las propiedades térmicas del objetivo, distorsiones geométricas y efectos radiométricos.

Figura 3.25: Termograma.

Debido a longitud de onda relativamente larga de la radiación térmica

(comparada con la radiación visible), la dispersión atmosférica es mínima. Sin embargo, la absorción por gases atmosféricos restringe normalmente la detección térmica a dos regiones específicas: 3-5 μm y 8-14 μm. Puesto que la energía decrece cuando la longitud de onda crece, los sensores térmicos tienen generalmente IFOVs amplios para asegurar que suficiente energía alcanza el detector para obtener una medida fiable. Por lo tanto, la resolución espacial de los sensores térmicos es bastante tosca, respecto a la resolución espacial disponible en las regiones visible e IR reflejado del espectro. Las imágenes térmicas pueden ser adquiridas durante el día o la noche (puesto que la radiación es emitida, no reflejada), y se usa para una variedad de aplicaciones tales como reconocimiento militar, control de catástrofes y monitorización de pérdidas de calor.

3.10 Distorsión geométrica en las imágenes

Cualquier imagen de teledetección, independientemente de si es adquirida por un escáner sobre un satélite, un sistema fotográfico sobre un avión, o cualquier otra combinación plataforma/sensor, tendrá varias distorsiones geométricas. Este problema es inherente a la teledetección, ya que intentamos representar exactamente superficies terrestres de tres dimensiones como una imagen de dos dimensiones. Todas las imágenes de teledetección son objeto de alguna forma de distorsión geométrica, dependiendo de la forma en que los datos son adquiridos. Estos errores pueden ser debidos a una variedad de factores, incluyendo uno o más de los siguientes:

- La perspectiva de la óptica del sensor
- El movimiento de los sistemas de exploración
- El movimiento e inestabilidad de la plataforma
- La altitud, orientación y velocidad de la plataforma
- El relieve del terreno
- La curvatura y rotación de la Tierra

Los sistemas de enfoque, tales como las cámaras utilizadas por la fotografía aérea, proporcionan vistas aéreas instantáneas (*snapshot*) de la Tierra. La principal distorsión geométrica de las fotografías aéreas verticales se debe al *desplazamiento del relieve* (ver figura 3.26). Los objetos situados justamente debajo del centro del sistema de lentes (óptica) de la cámara (nadir) tendrá visible solamente su parte superior, mientras que el resto de los objetos aparecerán inclinados desde el centro de la foto hacia fuera de forma que sus lados y parte superior son visibles. Si los objetos son altos o están alejados del centro de la foto, la distorsión y error posicional será más grande.

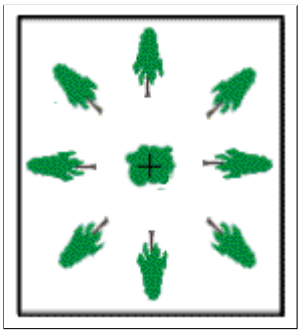


Figura 3.26: Desplazamiento del relieve.

Los sistemas de exploración de barrido presentan dos tipos principales de distorsión geométrica (ver figura 3.27). También exhiben desplazamiento del relieve (A), similar al de la fotografía aérea, pero solamente en la dirección paralela a la dirección de exploración. No hay desplazamiento directamente debajo del sensor, en el nadir. Como los sensores exploran a lo largo de la pasada, los lados y parte superior de los objetos son registrados y aparecen inclinados desde el punto nadir hacia fuera en cada línea de exploración. De nuevo, el desplazamiento crece a medida que nos desplazamos hacia el borde de la pasada. Otra distorsión (B) ocurre debido a la rotación de las ópticas de exploración. Como el sensor explora a lo largo de cada línea, la distancia del sensor al suelo crece más, lejos del centro de la pasada. Aunque el espejo de exploración gira a una velocidad constante, el IFOV del sensor se mueve más rápido (respecto al suelo) y explora un área más grande cuando se acerca a los bordes. Este efecto provoca la compresión de las características de la imagen en puntos fuera del nadir y se llama *distorsión de escala tangencial*.

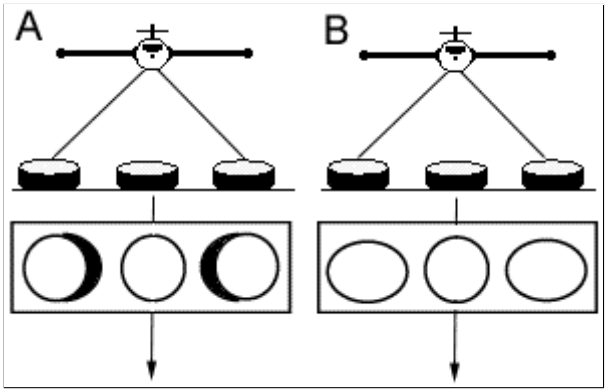


Figura 3.27: Distorsiones geométricas.

Todas las imágenes son susceptibles a las distorsiones geométricas provocadas por variaciones en la estabilidad de la plataforma incluyendo cambios en su velocidad, altitud y posición (orientación angular con respecto al suelo) durante la adquisición de los datos. Estos efectos son más pronunciados cuando utilizamos plataformas aéreas y se reducen en gran medida con el uso de plataformas de satélite, ya que sus órbitas son relativamente estables, particularmente en relación a su distancia de la Tierra. Sin embargo, la rotación hacia el este de la Tierra, durante una órbita del satélite provoca el barrido de los sistemas de exploración para cubrir un área ligeramente desplazada al oeste de cada exploración previa. Esto se conoce como *distorsión de sesgo* y es habitual en imágenes obtenidas desde un satélite con escáners multi-espectrales.

Las fuentes de la distorsión geométrica y del error posicional varían con cada situación específica, pero son inherentes en las imágenes de teledetección. En la mayoría de los casos, debemos ser capaces de eliminar, o al menos reducir estos errores, pero deben de tomarse en cuenta en cada caso antes de intentar realizar medidas o extraer información adicional.

3.11 Satélites meteorológicos

La monitorización y predicción climática fue una de las primeras aplicaciones civiles de la teledetección con satélites, que se inició con el primer satélite climático verdadero, TIROS-1 (Televisión and Infrared Observation Satellite),

lanzado en 1960 por los Estados Unidos. Varios satélites climáticos fueron lanzados durante los cinco años siguientes, en órbitas polares, proporcionando una cobertura periódica de los patrones meteorológicos globales. En 1966, la NASA (National Aeronautics and Space Administration) lanzó el satélite geoestacionario ATS-1 (Applications Technology Satellite) que proporcionó *imágenes hemisféricas* de la superficie terrestre y cubierta nubosa cada media hora (ver figura 3.28). Por primera vez, el desarrollo y movimiento de los sistemas climáticos podían ser monitorizados de forma rutinaria. Hoy en día, varios países gestionan satélites meteorológicos para monitorizar las condiciones climáticas alrededor del globo. Generalmente, estos satélites utilizan sensores que tienen una resolución espacial bastante grosera (comparados con los sistemas destinados a la observación del suelo) y proporcionan una gran cobertura aérea. Sus resoluciones temporales son generalmente bastantes altas, proporcionando observaciones frecuentes de la superficie terrestre, humedad atmosférica y cubierta nubosa, que permiten la monitorización continua de las condiciones climáticas globales, y por lo tanto la predicción. A continuación se describen dos de los satélites/sensores más representativos utilizados en aplicaciones meteorológicas.

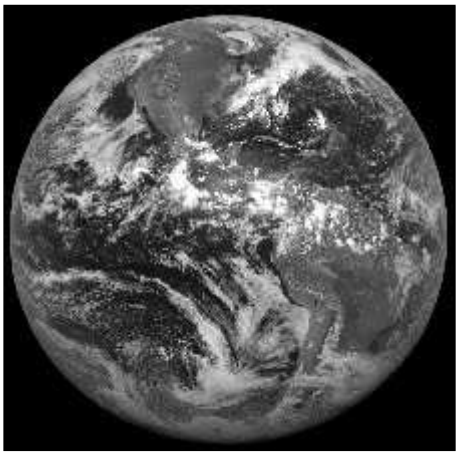


Figura 3.28: Imágenes hemisféricas.

GOES

El sistema que sigue a la serie ATS es el GOES (Geostationary Operational Environmental Satellite). Fueron diseñados por la NASA para la NOAA (National Oceanic and Atmospheric Adminstration) para proporcionar con frecuencia, imágenes a pequeña escala de la superficie terrestre y la cubierta nubosa. La serie GOES ha sido utilizada extensivamente por los meteorólogos para monitorizar y predecir el clima durante veinte años. Estos satélite son parte de la red global de satélite meteorológicos situados a intervalos de longitud de 70 grados aproximadamente alrededor de la Tierra para proporcionar una cobertura global.

Han sido lanzados dos generaciones de satélites GOES, cada una mide radiación emitida y reflejada de las que se pueden derivar la temperatura atmosférica, vientos, humedad y cubierta nubosa.

ATIROS

La NOAA es responsable también de otra serie de satélites que son útiles para usos meteorológicos, así como otras aplicaciones. Estos satélites, situados en órbitas polares (830-870 km sobre la Tierra) y sincronizados con el sol, son parte de la serie ATIROS (Advanced TIROS) y proporcionan información complementaria a los satélites meteorológicos geoestacionarios (tales como el GOES). Dos satélites, que proporcionan cobertura global, trabajan coordinadamente para asegurar que la antigüedad de los datos disponibles sobre cualquier región de la Tierra no supere las seis horas. Uno de los satélites cruza el ecuador de norte a sur al amanecer mientras que el otro lo cruza al atardecer.

El sensor primario a bordo de los satélites NOAA, utilizado tanto en meteorología como observación y reconocimiento de la Tierra a pequeña escala, es el AVHRR (Advanced Very High Resolution Radiometer). El sensor AVHRR detecta radiación en las porciones visible, IR cercano y medio e IR térmico del espectro electromagnético, sobre una área de 3000 km de ancho.

3.12 Satélites de observación terrestre

Landsat

Aunque muchos de los satélites meteorológicos (como los descritos en la sección anterior) también se utilizan para monitorizar la superficie de la Tierra, no están optimizados para una observación detallada. Impulsado por las excitantes perspectivas y el enorme éxito de los primeros satélites meteorológicos, así como también por las imágenes tomadas durante las misiones espaciales tripuladas, el primer satélite diseñado especialmente para monitorizar la superficie de la Tierra, el Landsat-1, fue lanzado por la NASA en 1972 (ver figura 3.29). Denominado inicialmente como ERTS-1 (Earth Resources Technology Satellite), Landsat fue diseñado como un experimento para probar la viabilidad de captar datos multi-espectrales de la Tierra desde una plataforma espacial no tripulada (satélite). Desde aquel momento, este exitoso programa ha captado una enorme cantidad de datos de la superficie terrestre desde varios satélites Landsat. Gestionados inicialmente por la NASA, la responsabilidad del programa Landsat fue transferido a la NOAA en 1983. En 1985, el programa pasó a proporcionar datos a usuarios y aplicaciones civiles.

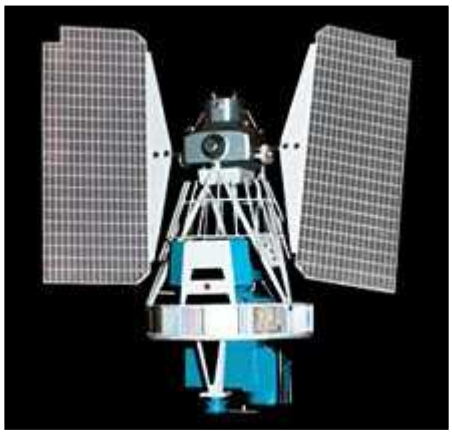


Figura 3.29: Landsat-1.

El éxito del Landsat se debe a varios factores, incluyendo: una combinación de sensores con bandas espectrales perfectamente ajustadas para la observación de la Tierra, una resolución espacial funcional y una buena

cobertura de la superficie (espacial y temporal). La elevada duración del programa ha proporcionado un voluminoso archivo de datos de recursos terrestre facilitando la monitorización, investigación y registros históricos durante un largo periodo de tiempo. Todos los satélites Landsat eran situados en órbitas polares, sincronizadas con el sol. Los primeros tres satélites Landsat están en altitudes que rondan los 900 km y tiene periodos de "nueva visita" de 18 días mientras que los últimos satélites están a 700 km de altitud aproximadamente y tienen periodos de "nueva visita" de 16 días. Todos los satélites Landsat cruzan el ecuador por la mañana para optimizar las condiciones de iluminación.

Un elevado número de sensores se han montado a bordo de la serie de satélites Landsat, incluyendo sistemas de cámaras RBV (Return Beam Vidicon), sistemas MSS (MultiSpectral Scanner) y TM (Thematic Mapper). El instrumento más popular de los primeros días del Landsat fue el MSS y más tarde el TM. Cada uno de estos sensores captaban datos sobre una superficie de aproximadamente 185 km de ancho.

SPOT

SPOT (Système Pour l'Observation de la Terre) es una serie de satélites de observación de la Tierra diseñada y lanzada por el CNES (Centre National d'Études Spatiales) de Francia, con ayuda de Suecia y Bélgica. SPOT-1 fue lanzado en 1986, con sucesores cada tres o cuatro años. Todos los satélites están situados en órbitas polares, sincronizadas con el sol y a altitudes que rondan los 830 km por encima de la Tierra, presentan periodos de "nueva visita" cada 26 días. Cruzan el ecuador alrededor de las 10:30 AM de la hora solar local. SPOT se ideó para ser un proveedor comercial de datos de la Tierra, y fue el primer satélite en utilizar tecnología de exploración de empuje.

Cada uno de los satélites SPOT tienen dos sistemas de observación HRV (High Resolution Visible) con idénticas características, que pueden operar independientemente y simultáneamente. Cada HRV es capaz de detectar un único canal de alta resolución espacial en modo pancromático (PLA), o tres canales con una resolución espacial ligeramente más pobre en modo

multi-espectral (MLA). Cada sensor HRV de exploración de empuje consta de cuatro arrays lineales de detectores: un array de 6000 elementos para el modo pancromático registra información con una resolución espacial de 10 m, y un array de 3000 elementos por cada banda espectral (tres bandas), registran la información con una resolución espacial de 20 m. El ancho de área cubierta en el nadir es en ambos modos de 60 km.

IRS

La serie de satélites IRS (Indian Remote Sensing), combinan las características de los sensores MSS/TM de Landsat y HRV de SPOT. El tercer satélite de la serie, IRS-1C, lanzado en diciembre de 1995 tiene tres sensores: una cámara de alta resolución de un sólo canal pancromático (PAN), un sensor LISS (Linear Imaging Self-Scanning Sensor) de cuatro canales con una resolución media y un sensor WiFS (Wide Field Sensor) de dos canales con una resolución relativamente baja.

Además de su alta resolución espacial, el sensor pancromático se puede orientar con ángulos de hasta 26 grados de exploración de barrido, facilitando la observación estereoscópica y reduciendo el periodo de "nueva visita" hasta los 5 días, similar al SPOT.

> **NOTA:** En el capítulo destinado a la descripción de la información (imágenes, datos colaterales, etc.) utilizados en la realización del proyecto, se realizará una descripción pormenorizada de cada uno de los sensores y de los datos proporcionados por esta serie de satélites. Concretamente, el cuarto satélite de la serie el IRS-1D al que pertenecen los datos utilizados en este proyecto.

3.13 Satélites de observación marina

Los océanos de la Tierra cubren más de las dos terceras partes de la superficie terrestre y juegan un papel importante en sistema climático global. También contienen una gran cantidad de organismos vivos y de recursos naturales sensibles a la polución y a peligros de origen humano. Los satélites/sensores

meteorológicos y de observación terrestre tratados en las dos secciones anteriores pueden utilizarse para monitorizar los océanos del planeta, pero hay otros sistemas satélite/sensor que han sido diseñados especialmente para este propósito.

El satélite Nimbus-7, lanzado en 1978, portaba el primer sensor, el CZCS (Coastal Zone Colour Scanner), proyectado específicamente para monitorizar los océanos de la Tierra. El principal objetivo de este sensor era observar el color y la temperatura del océano, concretamente las zonas costeras, con suficiente resolución espacial y espectral para detectar agentes contaminantes en lo niveles superiores del océano y determinar la naturaleza de los materiales suspendidos en el agua. El satélite Nimbus fue situado en una órbita polar, sincronizada con el sol, a una altitud de 955 km. Cruzaba el ecuador al mediodía local en la pasada ascendente y a la media noche local en la pasada descendente. El ciclo de órbitas permitía una cobertura global cada seis días, o cada 83 órbitas. El sensor CZCS consta de seis bandas espectrales, en la porción visible, IR cercana e IR térmico del espectro, y registra datos con una resolución espacial de 825 m en el nadir, sobre un área cubierta de 1566 km de ancho.

MOS

El MOS-1 (Marine Observation Satellite) fue lanzado por Japón en febrero de 1987 y fue seguido por su sucesor, el MOS-1b, en febrero de 1990. Estos satélites portaban tres tipos diferentes de sensores: un sensor MESSR (Multispectral Electronic Self-Scanning Radiometer) de cuatro canales, un sensor VTIR (Visible and Thermal Infrared Radiometer) de cuatro canales y un sensor MSR (Microwave Scanning Radiometer) de dos canales.

SeaWiFS

El SeaWiFS (Sea-viewing Wide-Field-of View Sensor) a bordo del transbordador SeaStar es un sensor de avanzado diseñado para monitorizar procesos oceánicos. Consta de ocho bandas espectrales de rangos de longitudes de onda muy estrechos ajustados para la detección y monitorización muy específica de varios fenómenos oceánicos, incluyendo: procesos de fitoplancton,

influencia del océano en los procesos climáticos y la monitorización de los ciclos del carbono, azufre y nitrógeno (ver figura 3.30). La altitud de la órbita es de 705 km y la hora local de cruce por el ecuador son las 12 PM. Hay dos combinaciones de resolución espacial y ancho del área cubierta para cada banda: un modo de resolución superior de 1.1 km (en el nadir) sobre un área cubierta de 2800 km de ancho, y un modo de resolución inferior de 4.5 km (en el nadir) sobre un área cubierta de 1500 km de ancho.

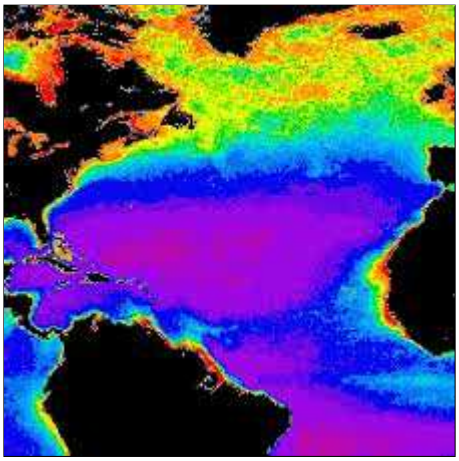


Figura 3.30: Concentraciones de fitoplancton.

3.14 Transmisión, recepción y procesamiento de los datos

Los datos obtenidos durante las misiones de teledetección aéreas pueden ser recuperados una vez que el avión aterrice. Entonces pueden procesarse y enviarse al usuario final. Sin embargo, los datos adquiridos desde los satélites necesitan ser transmitidos electrónicamente a la Tierra, puesto que el satélite continua en la órbita durante su tiempo de vida operacional. Las tecnologías desarrolladas para lograr esto pueden utilizarse también desde plataformas aéreas si los datos sobre la superficie se necesitan urgentemente.

Hay tres opciones principales para transmitir los datos adquiridos por el satélite a la Tierra (ver figura 3.31). Los datos pueden ser transmitidos a la Tierra directamente si una GRS (Ground Receiving Station) está en el

área de cobertura del satélite (A). Si este no es el caso, los datos pueden ser registrados a bordo del satélite (B) para transmitirlos a una GRS posteriormente. Los datos también pueden transmitirse a la GRS a través de un TDRSS (Tracking and Data Relay Satellite System) (C), que consta de una serie satélites de comunicación en órbitas geoestacionarias sincronizadas (geo-synchronous). Los datos se transmiten desde un satélite a otro hasta alcanzar el GRS apropiado.

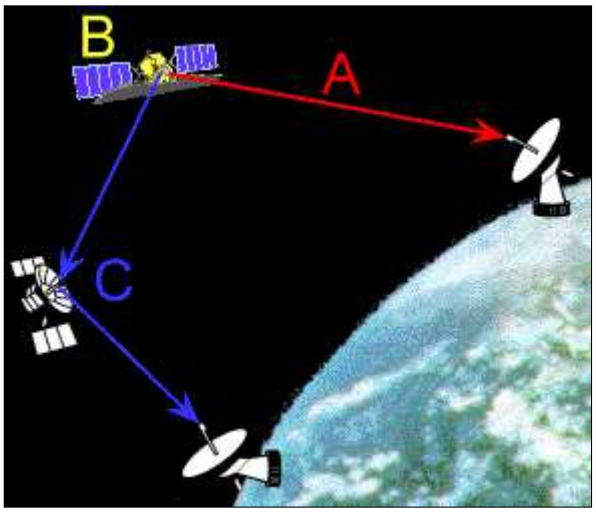


Figura 3.31: Transmisión de datos.

Los datos son recibidos en el GRS en formato RAW (en bruto). Entonces, si lo requieren, pueden ser procesados para corregir distorsiones sistemáticas, geométricas y atmosféricas, y ser trasladados a un formato estándar. Los datos se almacenan en algunos medios informáticos tales como cinta, disco o CD. Los datos son archivados típicamente en la mayoría de las estaciones de recepción y procesamiento, y completas librerías de datos para cada satélite/sensor son gestionadas por agencias gubernamentales así como compañías privadas.

Para muchos sensores es posible proporcionar imágenes *quick-look* (baja resolución) cuando se necesitan datos tan rápidamente como sea posible después de que estos hayan sido captados. Se utilizan sistemas de procesamiento muy rápidos para producir imágenes de baja resolución en papel (hard copy) o en formato digital (*soft copy*) a las pocas horas de la adquisición de los da-

tos. Las imágenes pueden ser transmitidas entonces por fax o digitalmente al usuario final. Una aplicación típica de este tipo de procesamiento rápido de datos es proporcionar imágenes de apoyo a la navegación en el Ártico, lo que permite entonces evaluar las condiciones actuales del hielo para tomar decisiones de navegación sobre la rutas más fáciles/seguras a través del hielo.

Las imágenes de baja resolución se usan para presentar las imágenes archivadas anteriormente puestas a la venta. La calidad espacial y radiométrica de este tipos de productos está degradada, pero son muy útiles para asegurar que la calidad global, cobertura y cubierta nubosa de los datos es la apropiada.

Capítulo 4

Interpretación y análisis

4.1 Introducción

Para aprovechar y hacer un buen uso de los datos de teledetección debemos ser capaces de extraer información significativa de las imágenes. La interpretación y análisis de las imágenes de teledetección implica la identificación y/o medida de varios objetivos en una imagen para extraer información útil sobre ellas. Los objetivos en las imágenes de teledetección pueden ser cualquier aspecto u objeto que puede ser observado en una imagen, y tiene las siguientes características:

- Los objetivos pueden ser un punto, línea o área. Esto significa que pueden tener cualquier forma, desde un autobús en un estacionamiento o una pista de aterrizaje.
- El objetivo debe ser distinguible en la imagen, debe contrastar con otras características de su alrededor.

Muchos de los procesos de interpretación e identificación en las imágenes de teledetección se realizan manualmente o visualmente (intérprete humano). En muchos casos esto se hace utilizando imágenes visualizadas en un formato gráfico o fotográfico, independiente del tipo de sensor usado para captar los datos.

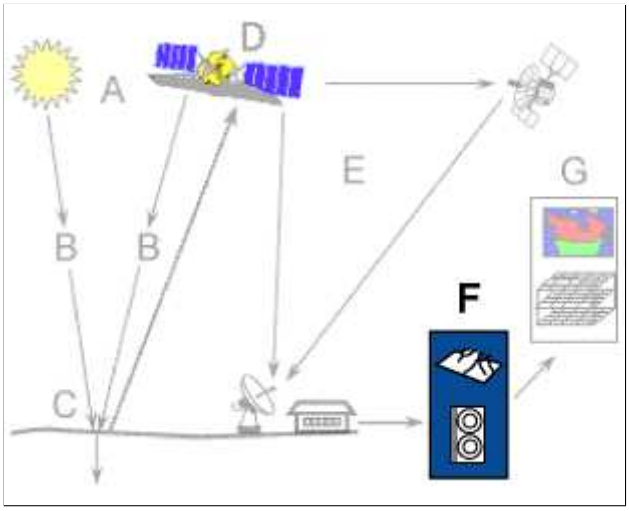


Figura 4.1: Interpretación y análisis.

Cuando los datos de teledetección están disponibles en formato digital (soft copy), se puede realizar un procesamiento y análisis digital utilizando un sistema informático. El procesamiento digital se puede realizar para mejorar los datos como un preludio para interpretación visual. El procesamiento y análisis digital puede llevarse a cabo para identificar automáticamente los objetivos y extraer información sin la intervención de un intérprete humano. Sin embargo, raras veces se realiza el procesamiento y análisis digital en sustitución de la interpretación manual. A menudo, se realiza para complementar y asistir al analista humano.

La interpretación y análisis manual data de los principios de la teledetección (interpretación de fotografías aéreas). El procesamiento y análisis digital es más reciente, favorecido con la llegada del registro digital de los datos de teledetección y el desarrollo de los ordenadores. Tanto las técnicas manuales como digitales para la interpretación de los datos de teledetección tienen sus ventajas y sus inconvenientes. Generalmente, la interpretación manual requiere poco, o casi ningún, equipo especializado. La interpretación manual esta limitada a menudo a analizar solamente un único canal de datos o imagen a la vez, debido a la dificultad de realizar la interpretación visual con varias imágenes. El entorno informático es más sensible a la manipulación de imágenes complejas, de varios canales o de varias fechas. En este

sentido, el análisis digital es útil para el análisis simultáneo de muchas bandas espectrales y para procesar grandes cantidades de datos mucho más rápido que un intérprete humano. La interpretación manual es un proceso subjetivo, es decir, que el significado de los resultados variará de un intérprete a otro. El análisis digital esta basado en la manipulación de los números digitales (píxeles) en un ordenador y, por lo tanto, es más objetivo, proporcionando generalmente unos resultados más consistentes. Sin embargo, determinar la validez y exactitud de los resultados del procesamiento digital puede ser difícil.

Es importante reiterar que el análisis visual y digital de los datos de teledetección no son mutuamente excluyentes. Ambos métodos tienen sus ventajas. En la mayoría de los casos, se emplea una combinación de ambos métodos para analizar las imágenes. De hecho, la última decisión sobre la utilidad y relevancia de la información extraída al final del proceso de análisis, aún debe ser tomada por humanos.

4.2 Elementos de la interpretación visual

Como mencionamos en la sección previa, el análisis de las imágenes de teledetección implica la identificación de varios objetivos en una imagen, y esos objetivos pueden ser características ambientales o artificiales que constan de puntos, líneas o áreas. Los objetivos pueden ser definidos en términos de la forma en la que ellos reflejan o emiten radiación.

¿Qué hace la interpretación de las imágenes más difícil que la interpretación visual de nuestro entorno? Por un lado, nosotros perdemos nuestro sentido de la profundidad cuando vemos una imagen de dos dimensiones, a menos que podamos verla *estereoscópicamente* para simular la tercera dimensión (altura). Actualmente, los beneficios de la interpretación es enorme en muchas aplicaciones cuando las imágenes son visualizadas en estéreo, ya que la visualización (y por tanto, el reconocimiento) de los objetivos se mejora notablemente. La observación directa de objetos desde arriba también proporciona un perspectiva muy diferente de la que estamos acostumbrados. La combinación de una perspectiva poco familiar con una escala diferente y falta de detalles reconocibles puede hacer irreconocibles en una imagen incluso los

objetos más familiares. Finalmente, nosotros solamente vemos las longitudes de onda de la región visible del espectro, la observación de longitudes de onda fuera de esta región del espectro es más difícil de comprender para nosotros.

El reconocimiento de objetivos es la clave para la interpretación y extracción de información. Observar las diferencias entre objetivos y fondo (background) implica la comparación de los diferentes objetivos, basada en alguno, o todos, de los elementos visuales: tono, forma, tamaño, patrón, textura, sombra y relación. La interpretación visual que utiliza estos elementos forma parte de nuestras vidas diarias, tanto si somos conscientes de ello o no. La identificación de objetivos en imágenes, basada en estos elementos visuales nos permite un mejor análisis e interpretación. La naturaleza de cada uno de estos elementos de interpretación se describe a continuación, junto con una imagen de ejemplo para cada uno.

El *tono* se refiere al brillo relativo o color de los objetos en una imagen (ver figura 4.2). Generalmente, el tono es el elemento fundamental para distinguir entre diferentes objetivos o características. Variaciones en el tono también permiten que elementos tales como la forma, textura o patrones de objetos sean distinguidos.

Figura 4.2: Tono.

La *forma* se refiere a la estructura general o silueta de objetos individuales (ver figura 4.3). La forma puede ser una característica muy interesante para la interpretación. Formas de bordes rectos representan típicamente objetivos urbanos o agrícolas (campos), mientras que características naturales, tales como bosques, son más irregulares en su forma, excepto donde el hombre ha construido una carretera o un cortafuego.

Figura 4.3: Forma.

El *tamaño* de los objetos en una imagen es una función de la escala (ver figura 4.4). Es importante evaluar el tamaño de un objetivo relativo a otros objetos de la escena, así como también el tamaño absoluto, para ayudar a la interpretación del objetivo. Una rápida aproximación del tamaño del objetivo puede conducir la interpretación a un resultado adecuado más rápidamente.

Figura 4.4: Tamaño.

El *patrón* se refiere a la disposición espacial de los objetos discernibles de forma visible (ver figura 4.5). Típicamente, una repetición ordenada de tonos y texturas similares producirá un patrón característico y a fin de cuentas reconocible. Un huerto con árboles uniformemente espaciados es un buen ejemplo de patrón.

Figura 4.5: Patrón.

La *textura* se refiere a la disposición y frecuencia de variación tonal en áreas particulares de una imagen (ver figura 4.6). Las texturas *gruesas* son regiones con una alta variación tonal (bosques, etc.), mientras que las texturas *suaves* tendrían una variación tonal muy pequeña (superficies asfaltadas, campos, etc.). La textura es uno de los elementos más importantes para distinguir características en imágenes.

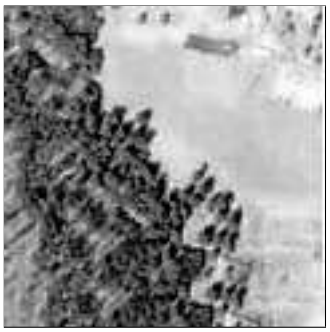


Figura 4.6: Textura.

La *sombra* también es útil en interpretación ya que puede proporcionar una idea del perfil y altura relativos de un objetivo u objetos facilitando la identificación (ver figura 4.7). Sin embargo, las sombras también pueden reducir o eliminar la posibilidad de interpretación en su área de influencia, ya que los objetivos afectados por las sombras son mucho menos (o nada) discernibles de sus entornos.

Figura 4.7: Sombra.

La *asociación* tiene en cuenta la relación entre objetos o características reconocibles en la proximidades del objetivo de interés. La identificación de características que uno esperaría encontrar en las proximidades del objetivo de interés, puede proporcionar información para facilitar su identificación. Por ejemplo, un lago está asociado con botes, una dársena de yates y un parque recreativo contiguo (ver figura 4.8).

Figura 4.8: Asociación.

4.3 Procesamiento de la imagen

En el mundo actual, donde la mayoría de los datos de teledetección se registran en formato digital, toda interpretación y análisis de imágenes involucra algún elemento de procesamiento digital. El procesamiento digital de imágenes puede implicar numerosos procedimientos incluyendo el formateo y corrección de los datos, la mejora digital para facilitar una mejor interpretación

visual, o incluso la clasificación automatizada de objetivos y características por un sistema informático. Para procesar imágenes de teledetección digitalmente, los datos deben estar registrados y disponibles en un formato digital adecuado para el almacenamiento en una cinta o disco informático. Obviamente, el otro requisito para el procesamiento digital de imágenes es un sistema informático, con el hardware y software apropiado para procesar los datos. Varios sistemas informáticos comerciales han sido desarrollados específicamente para el procesamiento y análisis de imágenes de teledetección.

La mayoría de las funciones disponibles en los sistemas de procesamiento y análisis digital de imágenes pueden ser clasificadas dentro de las siguientes cuatro categorías:

1. Preprocesamiento
2. Realce de la imagen
3. Transformación de la imagen
4. Clasificación de la imagen y análisis

Las *funciones de preprocesamiento* involucran aquellas operaciones que se requieren normalmente antes del análisis de los datos y de la extracción de información, y se agrupan generalmente en correcciones **radiométricas** y **geométricas**. Las *correcciones radiométricas* incluyen la corrección de las irregularidades del sensor y el ruido atmosférico, y la conversión de los datos para que de esta manera representen exactamente la radiación reflejada o emitida medida por el sensor. Las *correcciones geométricas* incluyen la corrección de las distorsiones geométricas debidas a las variaciones geométricas del par sensor-Tierra, y la conversión de los datos a coordenadas del mundo real (por ejemplo, latitud y longitud) sobre la superficie terrestre.

El objetivo del segundo conjunto de funciones de procesamiento, agrupado bajo el término de *mejora de la imagen*, es exclusivamente mejorar la apariencia de las imágenes para ayudar al análisis e interpretación visual (ver figura 4.9). Ejemplos de funciones de mejora incluyen la **expansión del contraste** para incrementar la diferencia tonal entre varias características de un escena, y el **filtrado espacial** para mejorar (o suprimir) patrones espaciales específicos de una imagen.

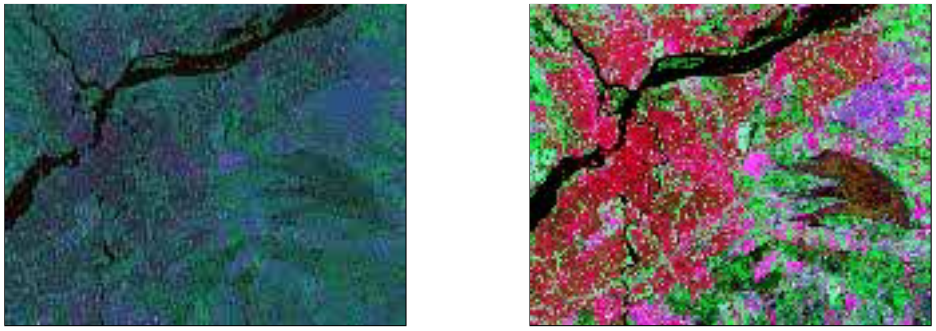


Figura 4.9: Mejora de la imagen.

Las *transformaciones de la imagen* son operaciones conceptualmente similares a las funciones de mejora de la imagen. Sin embargo, a diferencia de las operaciones de mejora de la imagen que son aplicadas normalmente a un único canal de datos a la vez, las transformaciones de la imagen implican habitualmente procesamiento combinado de datos de múltiples bandas espectrales. Operaciones aritméticas (por ejemplo, suma, resta, multiplicación y división) son realizadas para combinar y transformar las bandas originales en "nuevas" imágenes que mejoran o realzan ciertas características de la escena.

Las operaciones de *clasificación y análisis de la imagen* se utilizan para identificar y clasificar píxeles. La clasificación se realiza habitualmente sobre conjuntos de datos multicanal, asignando cada pixel de la imagen a una clase particular, atendiendo a las propiedades estadísticas del brillo de los píxeles (ver figura 4.10).

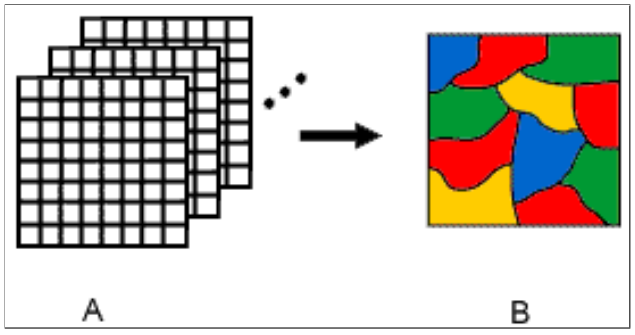


Figura 4.10: Clasificación y análisis.

Como comentamos en capítulos anteriores, la mayor parte de este proyec-

to se dedica al estudio y mejora de técnicas pertenecientes a la primera categoría (preprocesamiento), pero hemos creído conveniente incluir una breve descripción de todas y cada una de las operaciones destinadas al procesamiento de datos obtenidos en teledetección, con objeto de situar convenientemente el trabajo realizado.

4.4 Preprocesamiento

Las operaciones de preprocesamiento, denominadas a veces como *restauración* y *rectificación* de la imagen, se realizan para corregir distorsiones radiométricas y geométricas específicas de los datos. Las correcciones radiométricas pueden ser necesarias debido a las variaciones en la iluminación de la escena y la geometría de observación, condiciones atmosféricas, y la respuesta y ruido del sensor. Cada uno de estos variará dependiendo del sensor y plataforma utilizada para adquirir los datos y las condiciones durante su adquisición. También, puede ser deseable convertir y/o calibrar los datos para conocer la radiación (absoluta) o unidades de reflectancia para facilitar la comparación entre datos.

Las variaciones en la iluminación y la geometría de observación entre imágenes (para sensores ópticos) pueden ser corregidas mediante el modelado de la relación geométrica y la distancia entre el área de superficie terrestre observada, el sol y el sensor. Esto se requiere a menudo para poder comparar fácilmente imágenes captadas por diferentes sensores en fechas diferentes, o generar *mosaicos de imágenes* de un único sensor manteniendo uniformes las condiciones de iluminación de una escena a otra (ver figura 4.11).

Como vimos en el capítulo 2, la radiación se dispersa cuando pasa a través de la atmósfera e interactúa con ella. La dispersión puede reducir, o atenuar, en cierta medida la energía que ilumina la superficie. De igual forma, la atmósfera atenuará aún más, la energía que se propaga desde el objetivo hasta el sensor. Se pueden aplicar varios métodos de corrección atmosférica variando desde complejos métodos de modelado de las condiciones atmosféricas durante la adquisición de los datos, a simples cálculos basados solamente en los datos de las imágenes. Un ejemplo de este último método consiste en examinar los valores de brillo observados (números digitales), en un área

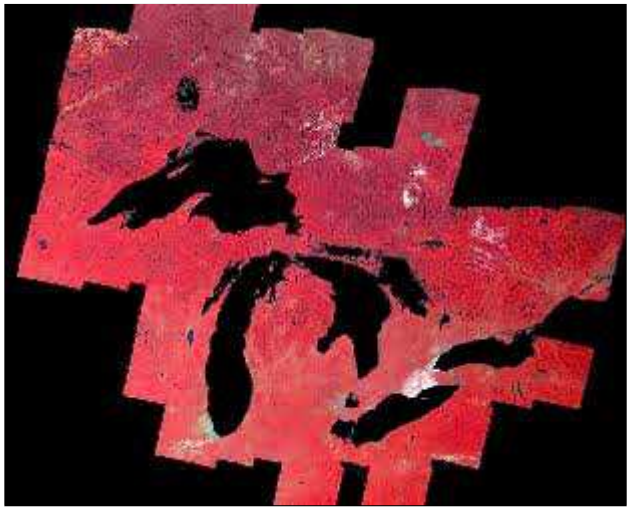


Figura 4.11: Mosaico de imágenes.

de sombra o para un objeto muy oscuro[1] (A) y determinar el valor mínimo (B) (ver figura 4.12). La corrección se aplica restando el valor mínimo observado, obtenido para cada banda específica, a todos lo píxeles de su banda correspondiente. Puesto que la dispersión depende de la longitud de onda, los valores mínimos variarán de una banda a otra. Este método supone que la reflectancia de estas características, si la atmósfera no afectase, debería ser muy pequeña o prácticamente cero. Si observamos valores mucho más grandes que cero, entonces se consideran que han resultado de la dispersión atmosférica.

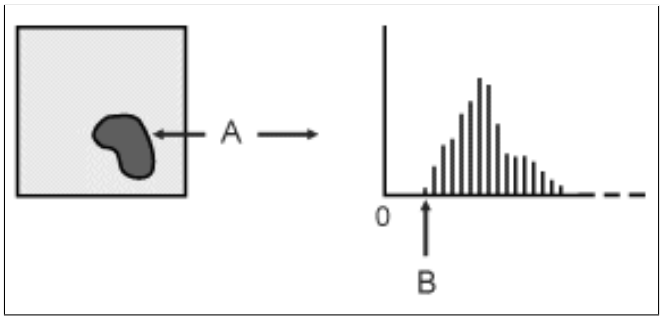


Figura 4.12: Corrección del efecto atmosférico.

El ruido de una imagen puede ser debido a irregularidades o errores que

[1] Áreas de baja reflectancia.

ocurren en la respuesta del sensor y/o el registro y transmisión de los datos. Las formas habituales de ruido incluyen el rayado sistemático o *bandeado* (*striping*) y las *líneas perdidas* (ver figuras 4.13 y 4.14). Ambos efectos deberían ser corregidos antes de realizar cualquier otro proceso (de mejora o clasificación).

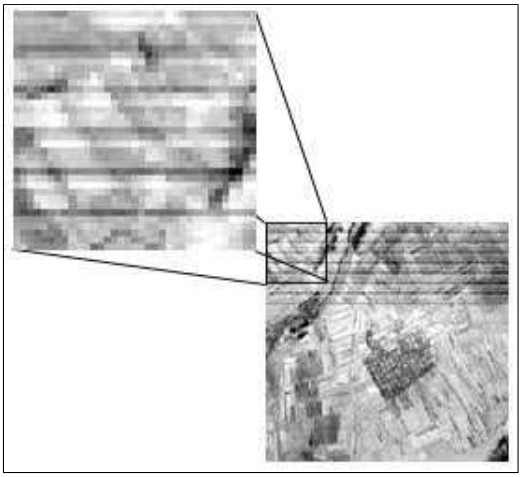


Figura 4.13: Bandeado (*striping*).

Figura 4.14: Líneas pérdidas.

El *bandeado* era común en los datos MSS de los primeros Landsat debido a las variaciones y deriva en la respuesta temporal de los seis detectores MSS. La deriva era diferente para cada uno de los seis sensores, provocando que el mismo brillo fuese presentado de forma diferente por cada detector. La apariencia global era por lo tanto la de una imagen con bandas. El proceso corrector hacía una corrección relativa entre los seis sensores para aproximar sus valores aparentes en cada línea.

La *pérdida de líneas* tiene lugar cuando en los sistemas se producen errores que provocan la pérdida de datos o datos defectuosos a lo largo de una línea de exploración. Las líneas pérdidas son "corregidas" normalmente reemplazando cada una de ellas con los valores de los píxeles situados en la línea superior o inferior, o con la media de los dos.

Para muchas aplicaciones cuantitativas de los datos de teledetección, es necesario convertir los número digitales (píxeles) a valores que representen la *reflectancia* o *emitancia* real de la superficie. Esto se hace basándose en conocimientos detallados de la respuesta del sensor y la forma en que la señal

analógica (es decir, la radiación reflejada o emitida) se convierte a números digitales (conversión analógica-digital). Resolviendo esta relación en sentido inverso, se puede calcular la radiancia absoluta para cada pixel, de forma que podamos realizar comparaciones con precisión de imágenes captadas por diferentes sensores en fechas diferentes.

En la sección 3.10 vimos que todas las imágenes de teledetección están afectadas de distorsiones geométricas. Estas distorsiones pueden deberse a varios factores, como: la perspectiva de la óptica del sensor, el movimiento del sistema de exploración, el movimiento de la plataforma, la altitud, posición y velocidad de la plataforma, el relieve del terreno, y la curvatura y rotación de la Tierra. Las correcciones geométricas se realizan para compensar estas distorsiones de forma que la representación geométrica de las imágenes se aproxime tanto como sea posible al mundo real. Muchas de estas variaciones son *sistemáticas* o *predecibles* en la naturaleza, y pueden considerarse durante un modelado exacto del movimiento del sensor y la plataforma, y de la relación geométrica de la plataforma con la Tierra. Otros errores *no sistemáticos* o *aleatorios*, no pueden ser modelados y corregidos de esta forma. Por lo tanto, debe realizarse el **registro geométrico** de las imágenes para conocer el sistema de coordenadas terrestre.

El proceso de registro geométrico implica identificar la fila y columna de varios puntos claramente discernibles, llamados *puntos de control terrestre* (GCP), en la imagen distorsionada (ver figura 4.15 - A), y hacerlos corresponder con sus posiciones reales en coordenadas terrestres (por ejemplo latitud, longitud). Las coordenadas terrestres reales se toman, normalmente, de un mapa (ver figura 4.15 - B), en papel o en formato digital. Esto es un registro imagen a mapa (*image-to-map*). Una vez que se han identificado varios pares de GCP bien distribuidos, la información de las coordenadas es procesada por el sistema informático para determinar las ecuaciones de transformación apropiadas y aplicarlas a las coordenadas de la imagen original (fila y columna) y hacerlas corresponder con sus nuevas coordenadas terrestres. El registro geométrico puede realizarse también mediante el registro de la imagen con respecto a otra imagen (o imágenes), en lugar de un mapa. Este registro se denomina registro imagen a imagen (*image-to-image*) y se rea-

liza a menudo antes de efectuar varios procedimientos de transformación o durante la comparación multi-temporal de imágenes.

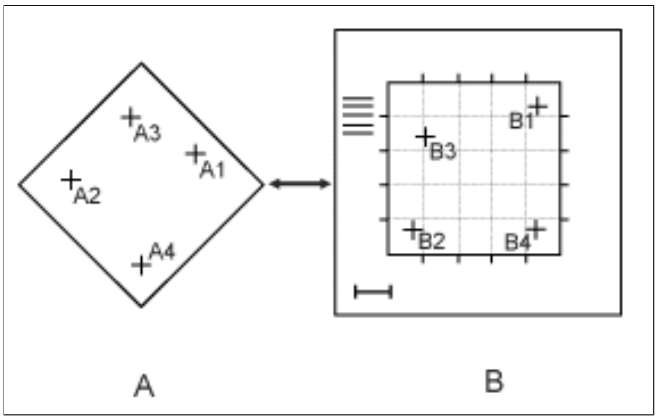


Figura 4.15: Proceso de registro geométrico.

Actualmente, para corregir geométricamente la imagen original distorsionada, se utiliza un procedimiento denominado *re-muestreo* (*re-sampling*) que tiene por objeto determinar los valores digitales a ubicar en las nuevas localizaciones de los píxeles en la imagen de salida corregida. El proceso de re-muestreo calcula el valor de los nuevos píxeles a partir del valor de los píxeles originales de la imagen sin corregir. Hay tres métodos para realizar el re-muestreo: el vecino más próximo, la interpolación bilinear y la convolución cúbica.

El *vecino más próximo* utiliza el valor digital del pixel en la imagen original que está más próximo a la nueva localización del pixel en la imagen corregida (ver figura 4.16). Este es el método más simple y no altera los valores originales, pero puede ocurrir que los valores de algún pixel se dupliquen mientras que otros se pierden. Este método tiende también a producir una imagen con apariencia "desarticulada" (o de líneas quebradas).

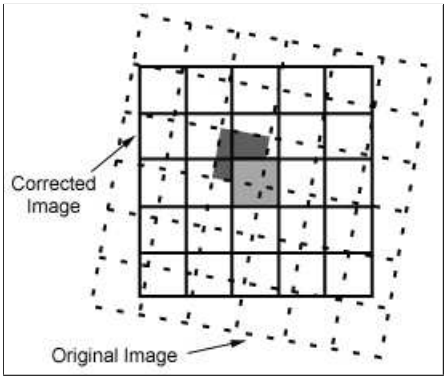


Figura 4.16: Vecino más próximo.

La *interpolación bilinear* asigna una media ponderada de los cuatro píxeles de la imagen original más próximos a la nueva localización del pixel (ver figura 4.17). Este proceso de promediado altera los valores de los píxeles originales y genera nuevos valores digitales en la imagen de salida. Esto puede resultar indeseable si se realizan posteriormente procedimientos tales como la clasificación basada en la respuesta espectral.

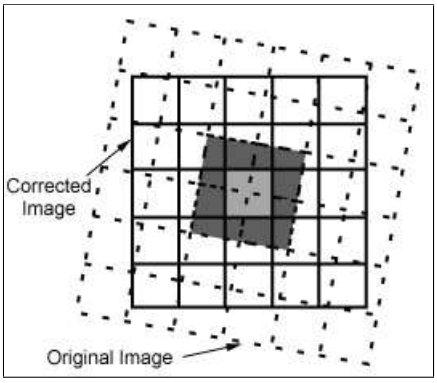


Figura 4.17: Interpolación bilinear.

La *convolución cúbica* va mucho más allá al calcular una media ponderada de dieciséis píxeles de la imagen original que circundan la localización del nuevo pixel (ver figura 4.18). Como ocurría con la interpolación bilinear, este método genera nuevos valores en la imagen de salida.

Sin embargo, estos dos últimos métodos producen imágenes con un mejor aspecto visual y evita la apariencia "desarticulada" del método del vecino más próximo.

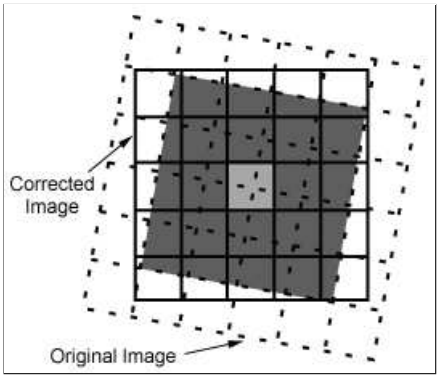


Figura 4.18: Convolución cúbica.

4.5 Realce de la imagen

Las mejoras se realizan para facilitar la interpretación visual y la comprensión de las imágenes. La ventaja de las imágenes digitales es que nos permite manipular los valores digitales de los píxeles. Aunque se pueden hacer ajustes radiométricos para corregir la iluminación, los efectos atmosféricos y las características del sensor antes de su distribución, la imagen puede aún no estar optimizada para la interpretación visual. Los dispositivos de teledetección, particularmente aquellos que operan desde plataformas de satélite, deben diseñarse para afrontar niveles de energía objetivo/fondo típicos, niveles que se corresponden con las condiciones que habitualmente se presentarán en su tiempo de vida operacional. Con las grandes variaciones en la respuesta espectral de diversos conjuntos de objetivos (por ejemplo bosques, desiertos, agua, etc.) ninguna corrección radiométrica genérica podría considerar y visualizar de forma óptima el rango de brillo y el contraste para todos ellos. Así, para cada aplicación y cada imagen, se necesita, en la mayor parte de los casos, un ajuste individual (o específico) del rango y distribución del brillo.

En imágenes RAW (en bruto), los datos útiles a menudo ocupan solamente una pequeña porción de rango de valores digitales disponible (habitualmente 256 niveles = 8 bits). La mejora del contraste implica cambiar los valores originales ya que se utiliza una mayor porción del rango disponible, por esa razón incrementamos el contraste entre objetivos y su fondo (teóricamente). La clave para comprender las mejoras del contraste es entender el

concepto de *histograma* de la imagen. Un *histograma* es una representación gráfica de los valores de brillo que contiene un imagen (ver figura 4.19). Los valores de brillo (habitualmente 256 niveles = 8 bits) se representan sobre el eje de abscisas del gráfico. La frecuencia de ocurrencia de cada uno de estos valores en la imagen se representan sobre el eje de ordenadas.

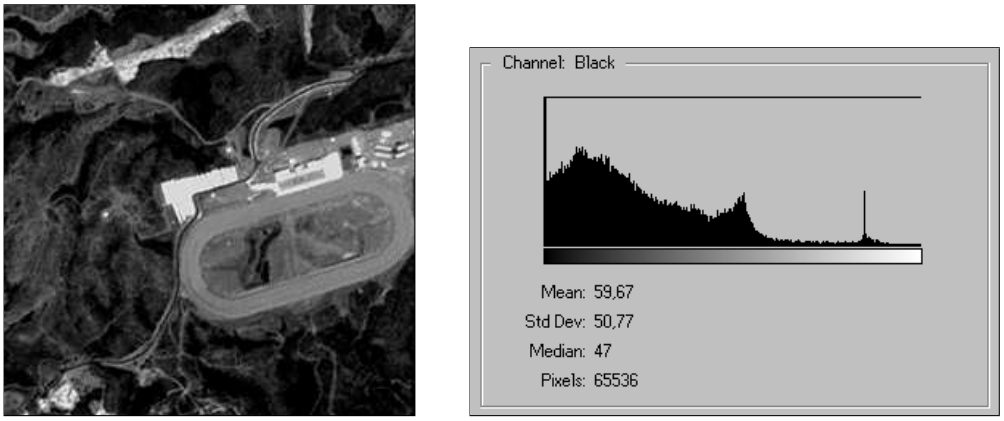


Figura 4.19: Histograma de la imagen.

Mediante la manipulación del rango de valores digitales de una imagen, representado gráficamente por su histograma, podemos aplicar varias mejoras a los datos. Hay diferentes métodos para mejorar el contraste y el nivel de detalle en una imagen. El tipo de mejora más simple consiste en un *estiramiento lineal del contraste* (ver figura 4.20).

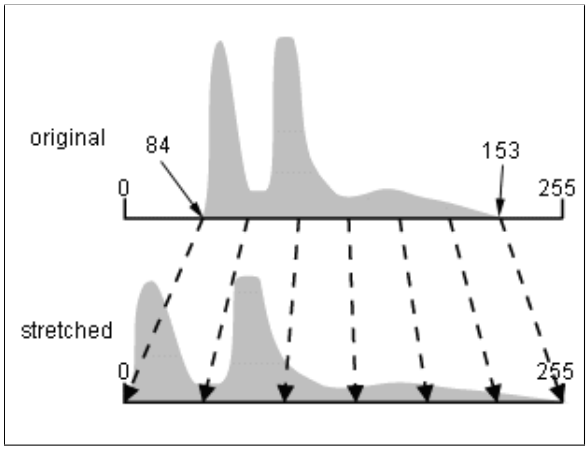


Figura 4.20: Estiramiento lineal del contraste.

Este método implica identificar los extremos del histograma (normalmente el valor máximo y mínimo de los valores de brillo de la imagen) y aplicar una transformación para estirar este rango y ocupar el rango completo. Un estiramiento lineal expande uniformemente un pequeño rango para cubrir el rango completo de valores. Esta mejora del contraste de la imagen, donde las zonas claras aparecen más claras y las oscuras aparecen más oscuras, facilita la interpretación visual.

Una distribución uniforme del rango de valores de entrada a lo ancho del rango disponible, puede no ser siempre una mejora adecuada, especialmente si el rango de entrada no está uniformemente distribuido. En este caso, puede ser mejor realizar una *ecualización del histograma* (ver figura 4.21). Esta técnica asigna más valores (rango) a las regiones del histograma que presentan una mayor frecuencia (mejora global). De esta forma, los detalles de estas áreas estarán más realzados respecto a los de aquellas áreas del histograma original que presentan una menor frecuencia. En otros casos, puede ser deseable mejorar el contraste sólo en una región específica del histograma (mejora local).

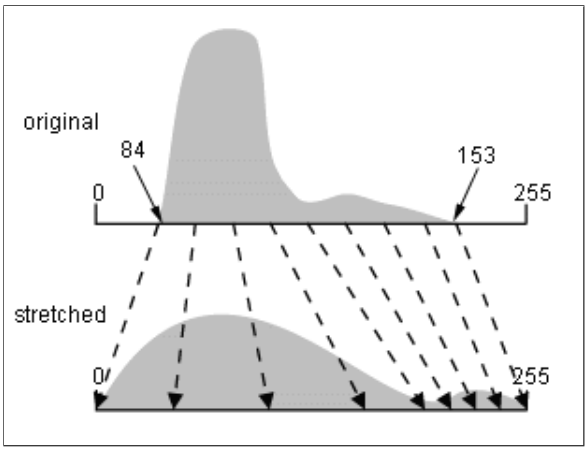


Figura 4.21: Ecualización del histograma.

El filtrado espacial abarca otro conjunto de funciones de procesamiento digital que se utilizan para mejorar la apariencia de una imagen. Los filtros espaciales se diseñan para realzar o suprimir características específicas de una imagen basados en su relación espacial. La *frecuencia espacial* esta relacio-

nado con el concepto de *textura* y se refiere a la frecuencia de las variaciones de tono que aparecen en una imagen. Las áreas de textura "gruesa" de una imagen, donde los cambios en el tono sobre una pequeña área son abruptos, tienen una alta frecuencia espacial, mientras que las áreas de textura "fina", con pequeñas variaciones en el tono sobre una pequeña área, tienen frecuencias espaciales bajas. Un *procedimiento de filtrado* común implica mover una "ventana" de unos pocos píxeles de dimensión (por ejemplo 3×3, 5×5, etc.) sobre cada pixel de la imagen, aplicando un cálculo matemático que utiliza los valores de los píxeles situados debajo de esa ventana, y sustituye el pixel central con el nuevo valor (ver figura 4.22). Esta ventana se mueve a lo largo de la imagen pixel a pixel, tanto en filas como en columnas, y el cálculo se repite hasta que la imagen completa ha sido filtrada y se ha generado una "nueva" imagen. Modificando el cálculo realizado y los pesos de los píxeles individuales de la ventana del filtro, se pueden diseñar filtros para realzar o suprimir diferentes tipos de características.

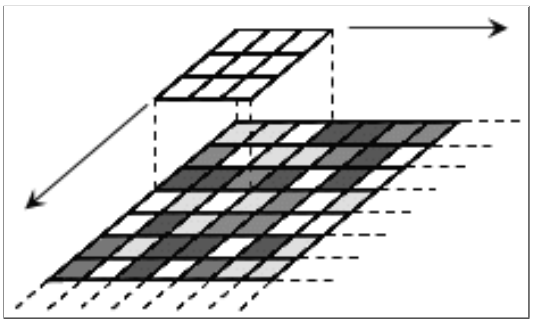


Figura 4.22: Procedimiento de filtrado.

Los *filtros pasa bajas* se diseñan para enfatizar áreas grandes y homogéneas de tonos similares y reducir los detalles más pequeños de la imagen. Así, los filtros pasa bajas sirven generalmente para suavizar la apariencia de una imagen. Los filtros de la media y la mediana, utilizados a menudo con imágenes de radar, son ejemplos de filtros pasa bajas (ver figuras 4.23 y 4.24).

Los *filtros pasa altas* hacen lo contrario y sirven para resaltar la apariencia de los detalles finos de una imagen. Una implementación de un filtro pasa bajas aplica primero un filtro pasa bajas a una imagen y entonces resta el

Figura 4.23: Imagen sin filtrar.

Figura 4.24: Imagen filtrada pasa bajas.

resultado a la imagen original, quedando solamente la información de las altas frecuencias espaciales. Los filtros direccionales, o detectores de bordes se diseñan para realzar las características lineales, tales como carreteras o caminos (ver figuras 4.25 y 4.26).

Figura 4.25: Imagen sin filtrar.

Figura 4.26: Imagen filtrada pasa altas.

Estos filtros también pueden diseñarse para mejorar características orientadas en direcciones específicas y son útiles por ejemplo, para la detección de estructuras lineales en geología.

4.6 Transformación de la imagen

Las transformaciones de la imagen implican generalmente la manipulación de múltiples bandas de datos, tanto de una única imagen multi-espectral o de dos o más imágenes de la misma área adquiridas en diferentes fechas (serie multi-temporal de imágenes). De cualquier forma, las transformaciones de la imagen generan "nuevas" imágenes de dos o más fuentes que realzan características particulares o propiedades de interés, mejor que las imágenes de entrada originales.

Las transformaciones de imagen básicas aplican simples operaciones aritméticas a las imágenes. La *diferencia de imágenes* se usa a menudo para identificar cambios que han ocurrido entre imágenes recogidas en diferentes fechas (registradas geométricamente). Escalando la imagen resultado mediante la suma de una constante (en la mayoría de los casos 127) a los valores de salida obtendremos una imagen "diferencia" adecuada (ver figura 4.27). En esta imagen, las áreas donde había pocos o ningún cambio entre las imágenes originales, tendremos valores de brillo próximos a 127 (niveles de gris medios), mientras que aquellas áreas donde tienen lugar cambios significantes tendrán valores más altos o bajos que 127 (más claros u oscuros dependiendo de la "dirección" del cambio de reflectancia entre las dos imágenes). Este tipo de transformación puede ser útil para detectar cambios urbanos y para identificar áreas donde se está produciendo deforestación.

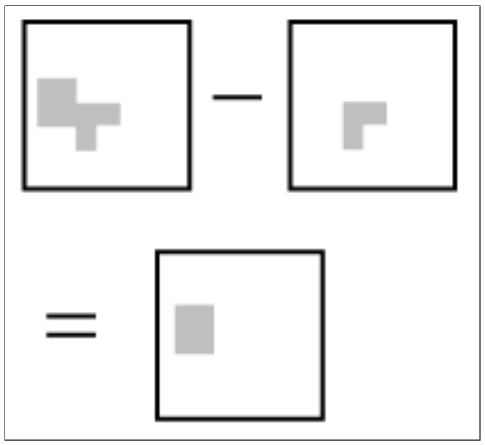


Figura 4.27: Diferencia de imágenes.

La *división de imágenes* o cociente espectral es una de las transformaciones más comunes aplicadas a las imágenes. El cociente de la imagen sirve para realzar variaciones sutiles en la respuesta espectral de las cubiertas de varias superficies. Dividiendo los datos de dos bandas espectrales diferentes, la imagen resultado acentúa las variaciones en las pendientes de las curvas de reflectancia espectral entre los dos rangos espectrales que de otra forma estarían enmascaradas por las variaciones de brillo en cada una de las bandas (ver figura 4.28).

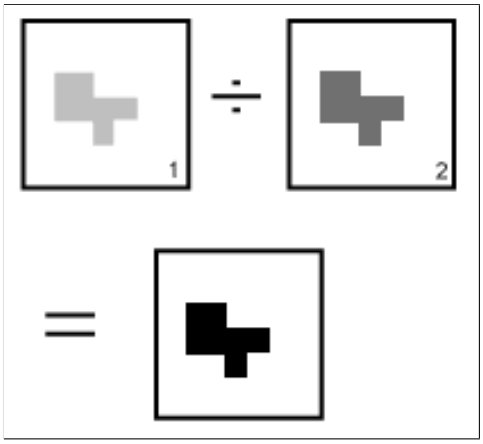


Figura 4.28: División de imágenes.

Otra ventaja del cociente espectral es que, ya que estamos considerando valores relativos (es decir porcentajes) en lugar de valores de brillo absolutos, las variaciones en la iluminación de la escena como resultado de los efectos topográficos se reducen. Así, aunque las reflectancias absolutas para un bosque situado en pendiente puede variar dependiendo de su orientación respecto a iluminación solar, el porcentaje de su reflectancias entre las dos bandas debería ser siempre muy similar. Porcentajes más complejos que involucran la suma y diferencia entre bandas espectrales para varios sensores, se han desarrollado para monitorizar las condiciones de la vegetación. Una transformación de la imagen ampliamente utilizada es NDVI (Normalized Difference Vegetation Index) que ha sido utilizada para monitorizar las condiciones de la vegetación a escalas continentales y globales utilizando el sensor AVHRR (Advanced Very High Resolution Radiometer) a bordo de la

serie de satélites NOAA (ver figura 4.29).

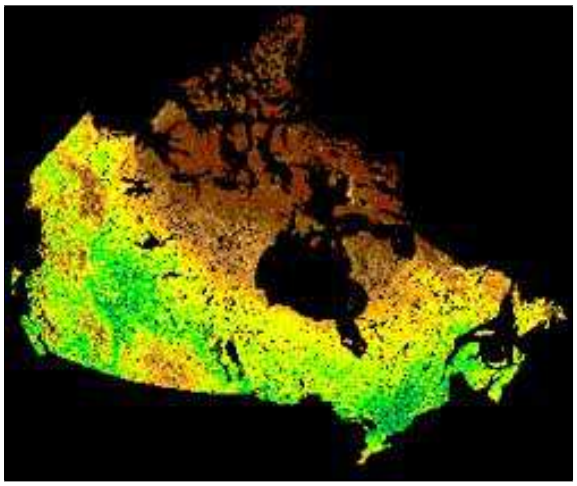


Figura 4.29: Índice de vegetación de diferencia normalizada.

Las diferencias de bandas de datos multi-espectrales están a menudo altamente correlacionadas y, por lo tanto, contienen información similar. Por ejemplo, las bandas Landsat MSS 4 y 5 (verde y roja, respectivamente) tienen habitualmente una apariencia visual similar ya que las reflectancias para la misma superficie cubierta son casi iguales. Las técnicas de transformación de la imagen basadas en procesamientos complejos de las características estadísticas de conjuntos de datos multi-banda pueden utilizarse para reducir esta redundancia de datos y la correlación entre bandas. Una de estás transformaciones se denomina *análisis de componentes principales* (ver figura 4.30). El objetivo de esta transformación es reducir el número de bandas en los datos, y comprimir mucha de la información de la bandas originales en pocas bandas. Las "nuevas" bandas que resultan de este procedimiento estadístico se denominan componentes. Este proceso intenta maximizar (estadísticamente) la cantidad de información (o varianza) de los datos originales dentro de un número mínimo de componentes. Como ejemplo de uso del análisis de componentes principales, un conjunto de siete bandas de datos TM (Thematic Mapper) puede ser transformado de modo que las tres primeras componentes principales contienen aproximadamente un 90% de de la información de las siete bandas originales. La interpretación y análisis de estas tres bandas de datos, combinándolas visualmente o digitalmente, es más simple y eficiente

que intentar utilizar las siete bandas disponibles. El análisis de las componentes principales, y otras transformaciones complejas, pueden utilizarse como técnica de realce para mejorar la interpretación visual o para reducir el número de bandas a utilizar como entrada en procedimientos digitales de clasificación.

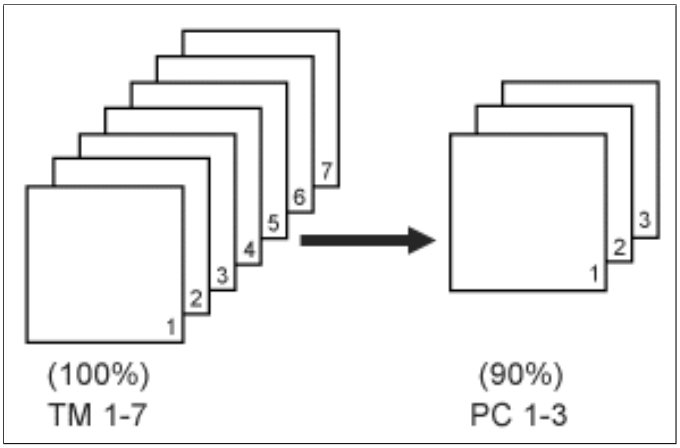


Figura 4.30: Análisis de las componentes principales.

4.7 Clasificación y análisis de la imagen

Un analista humano intentando clasificar características en un imagen utiliza los elementos de la interpretación visual para identificar grupos homogéneos de píxeles que representan objetivos o clases de interés. La clasificación digital de imágenes utiliza información espectral representada por los números digitales de una o más bandas espectrales, e intenta clasificar cada pixel individual basado en esta información espectral (ver figura 4.31). Este tipo de clasificación se denomina *reconocimiento de patrones espectrales*. En cualquier otro caso, el objetivo es asignar todos los píxeles de la imagen a clases particulares o temas (por ejemplo, agua, bosque conífero, bosque caduco, zonas urbanas, etc.). La imagen resultado clasificada consta de un mosaico de píxeles, cada uno de ellos perteneciente a un tema particular, y es esencialmente un "mapa" temático de la imagen original.

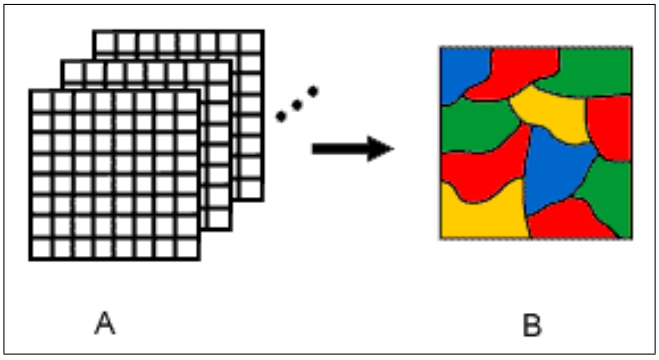


Figura 4.31: Clasificación y análisis.

Cuando hablamos de clases, necesitamos distinguir entre clases de *información* y clases *espectrales*. Las clases de *información* son aquellas categorías de interés que el analista está realmente intentando identificar en las imágenes, tales como diferentes clases de cultivos, diferentes tipos de bosques, etc. Las clases *espectrales* son grupos de píxeles que son uniformes con respecto a sus valores de brillo en los diferentes canales espectrales de los datos. El objetivo es ajustar las clases espectrales de los datos a las clases de información de interés. Rara vez hay un ajuste uno a uno entre estos dos tipos de clases. Más bien, clases espectrales únicas pueden no corresponderse necesariamente con alguna clase de información de interés para el analista. En otro caso, una clase de información de gran extensión (por ejemplo, un bosque) puede contener varias subclases espectrales con variaciones espectrales únicas. Siguiendo con el ejemplo del bosque, las subclases espectrales puede ser debidas a variaciones en la edad, especie y densidad, o quizás como resultado de sombras o variaciones en la iluminación de la escena. Es trabajo del analista decidir sobre la utilidad de las diferentes clases espectrales y su correspondencia con clases de información.

Los procedimientos de clasificación habituales pueden dividirse en dos grandes categorías basadas en el método utilizado: clasificación *supervisada* y clasificación *no supervisada*. En una clasificación *supervisada*, el analista identifica en la imágenes muestras representativas homogéneas de los diferentes tipos de superficies de interés (clases de información). Estas muestras se denominan *áreas de entrenamiento*. La selección de las áreas de entrenamiento apropiadas se basa en el experiencia del analista con el área geográfica

y su conocimiento de los diferentes tipos de superficies presentes en la imagen. Así, el analista esta "supervisando" la categorización de un conjunto específico de clases. La información numérica en todas la bandas espectrales de los píxeles que comprenden estas áreas se utiliza para "entrenar" un algoritmo para reconocer espectralmente áreas similares para cada clase. El ordenador utiliza un algoritmo, para determinar las "firmas" espectrales de cada clase de entrenamiento. Una vez el algoritmo ha determinado las firmas para cada clase, cada pixel de la imagen es comparado con estas firmas y etiquetado como perteneciente a la clase que más se le parece digitalmente. Así, en una clasificación supervisada, primero identificamos las clases de información que después son utilizadas para determinar las clases espectrales que las representan (ver figura 4.32).

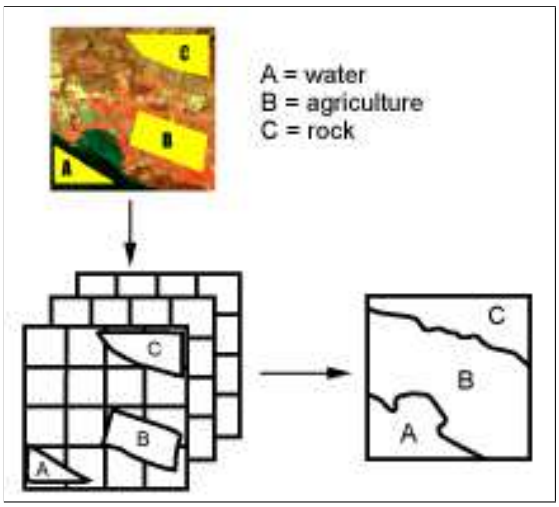


Figura 4.32: Clasificación supervisada.

La clasificación *no supervisada* invierte en esencia el proceso de clasificación supervisada. Las clases espectrales son agrupadas primero, basadas solamente en la información numérica de los datos, y son asignadas por el analista entonces a clases de información (si es posible). Algoritmos de agrupamiento o *clustering* se utilizan para determinar estadísticamente agrupamientos o estructuras naturales en los datos. Habitualmente, el analista especifica cuantos grupos o *clusters* se buscan en los datos. Además para especificar el número de clases deseado, el analista también puede especificar paráme-

tros asociados a la distancia de separación entre los grupos y la variación dentro de cada grupo. El resultado final de este proceso de agrupamiento iterativo puede proporcionar algunos grupos que el analista querrá combinar posteriormente, o grupos que deberían dividirse más adelante (esto requiere una aplicación posterior del algoritmo de agrupamiento). De este modo, la clasificación no supervisada no está libre completamente de la intervención humana. Sin embargo, no comienza con un conjunto predeterminado de clases de información como ocurre en la clasificación supervisada (ver figura 4.33).

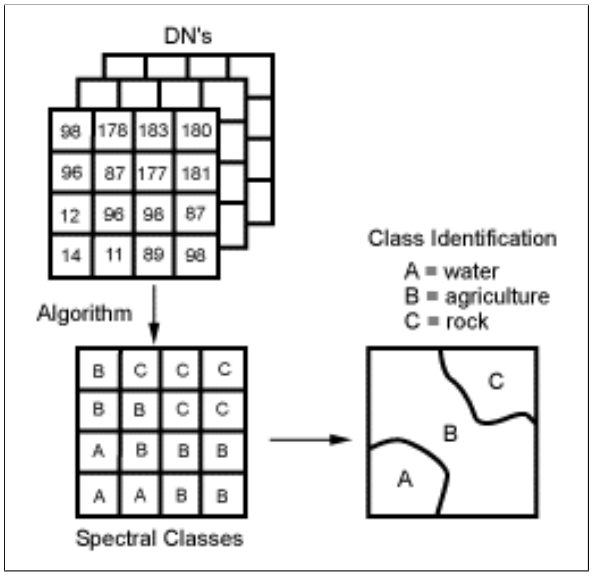


Figura 4.33: Clasificación no supervisada.

Parte II

Detección de cambios urbanos

Capítulo 5

Datos

5.1 Introducción

En la sección 1.5 comentamos brevemente algunos detalles relacionados con los datos utilizados en este proyecto. En secciones posteriores entraremos en profundidad en aspectos de especial importancia a la hora de seleccionar los datos a utilizar en una aplicación como la que nos ocupa: la detección de cambios urbanos a partir de imágenes de satélite.

> **NOTA:** El término "datos" hace referencia tanto a las imágenes como la información que las acompaña, información que resulta esencial para su posterior utilización, y que está relacionada con aspectos geográficos, radiométricos, etc. de las imágenes.

5.2 Satélite IRS-1D

Los datos empleados en el desarrollo del proyecto han sido proporcionados por Euromap GmbH[1], empresa distribuidora en Europa de los datos generados por los satélites de la serie IRS (Indian Remote Sensing).

El satélite IRS-1D fue lanzado con éxito, el 27 de septiembre de 1997 por una lanzadera PSLV y activado a mediados de octubre de 1997. Su principal objetivo es adquirir, de forma sistemática y repetitiva, datos de la superficie terrestre en condiciones de iluminación constantes. El satélite opera en una

[1] `http://www.euromap.de`

órbita polar (circular), sincronizada con el sol. Sus parámetros orbitales son los siguientes:

Parámetro	**Valor**
Órbitas/Periodo	341 órbitas
Periodo	24 días
Altitud*	817 km
Eje semi-mayor	7195.11 km
Inclinación	98.69º
Excentricidad	0.001
Periodo	101.35 min
Distancia entre órbitas adyacentes	117.5 km
Velocidad de cobertura	6.65 km/s

*En la pasada descendente

Tabla 5.1: Parámetros orbitales del satélite IRS-1D.

5.2.1 Sensores

Los satélites de la serie IRS disponen, como comentamos en la sección 3.12, de tres sensores: una cámara PAN (PANchromatic) de alta resolución (5.8 m) de un sólo canal, un sensor LISS-III (Linear Imaging Self-Scanning Sensor) de resolución media (23.5-70.5 m) de cuatro canales y un sensor WiFS (Wide Field Sensor) de baja resolución (188.3 m) de dos canales.

La cobertura del sensor LISS-III en las banda VIS es de 141 km de ancho mientras que en la banda SWIR es de 148 km. La cobertura de los sensores PAN y WiFS es de 70 km y 810 km de ancho respectivamente. La tabla 5.2 muestra detalles relacionados con la superposición (*overlap*) y separación (*sidelap*) entre escenas.

> **NOTA:** El *overlap* y *sidelap* son esenciales en aspectos tales como: la creación de mosaicos, cálculo del periodo de "nueva visita", etc.

Sensor	**Resolución (m)**	**Cobertura (km×km)**	*Overlap* **(km)**	*Sidelap** **(km)**
LISS-III				
VIS	23.5	141×141	7	23.5
SWIR	70.5	141×148	7	30
PAN	5.8	70×70	2	1
WiFS	188.3	810×810	80%	85%

*En el ecuador

Tabla 5.2: *Overlap*/*Sidelap* entre escenas.

Según lo expuesto anteriormente, el sensor que ofrece una resolución espacial lo suficientemente buena como para atacar con solvencia el problema, es la cámara PAN (PANchromatic). La tabla 5.3 muestra las características técnicas de esta cámara.

Característica	**Valor**
Resolución espacial	5.8 m
Resolución radiométrica	6 bits
Resolución espectral	500-750 nm
Cobertura-ancho	63-70 km
Distancia focal	974.8 mm
Arrays de CCDs	3 arrays ×4096 CCDs
Tamaño CCD	7 μm ×7 μm
Tiempo de integración	0.8836458 ms

Tabla 5.3: Características técnicas de la cámara PAN.

Como podemos observar, la cámara PAN proporciona una resolución espacial (5.8 m), radiométrica[2] (6 bits = 64 niveles de gris), espectral (banda VIS) y temporal (24 días) adecuadas. Lógicamente, cuanto mayor sea la resolución espacial de las imágenes utilizadas, mejores serán los resultados obtenidos. Sin embargo, el número de satélites comerciales capaces de proporcionar imágenes con resoluciones superiores a 5 m son muy pocos y con-

[2]La resolución espectral de una cámara PAN es de 6 bits, sin embargo los productos suministrados por Euromap GmbH presentan una resolución espectral de 8 bits (*histogram stretching*).

secuentemente, a un precio muy elevado. Personalmente, me atrevería a fijar el límite de este parámetro para aplicaciones de detección de cambios como la que nos ocupa en los 5 m, donde un cambio de un pixel se corresponde con un área de aproximadamente 5 m^2, resoluciones inferiores a los 5 metros harían inviables este tipo de aplicaciones. Esto da una idea de la precisión con la que se tiene que realizar la corrección geométrica, ya que un desajuste de tan sólo un pixel provocaría la aparición de cambios indeseados e irreales.

5.2.2 Path/Row

El siguiente paso consistiría en determinar la región geográfica de interés, en este caso, el término municipal de Málaga[3]. Llegado este punto, es necesario identificar la órbita de interés y el punto de adquisición, denominados *path* y *row* respectivamente. Para ello, el propio proveedor, en este caso Euromap GmbH, proporciona mapas path/row y calendarios de adquisición. En la figura 5.1 podemos observar un mapa path/row para Andalucía oriental. En este mapa podemos identificar claramente el path/row correspondiente al término municipal de Málaga.

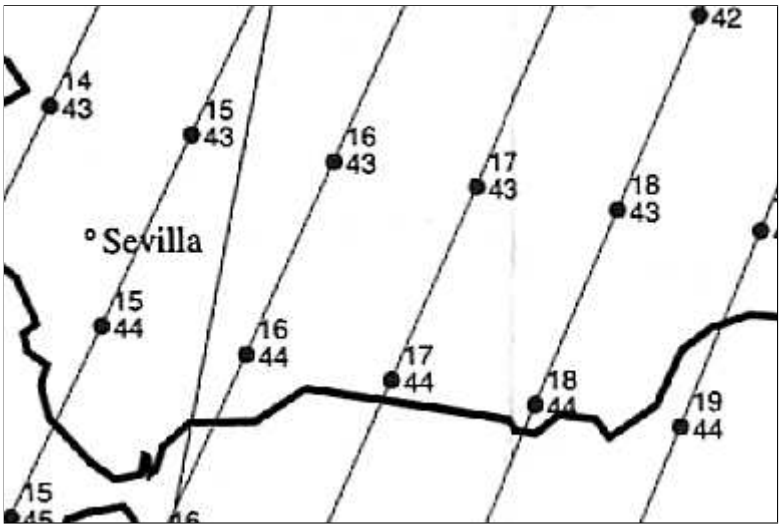


Figura 5.1: Mapa path/row para Andalucía oriental.

Las sucesivas órbitas abarcan una distancia de 2820.5 km en el ecuador.

[3]Coordenadas geográficas: 4º 25' W - 36º 43' N.

La figura 5.2 muestra un patrón típico de órbitas del IRS-1D. Durante el periodo de 24 días, el satélite recorre la totalidad de la superficie terrestre en 341 órbitas, entre los 81° N y 81° S de latitud.

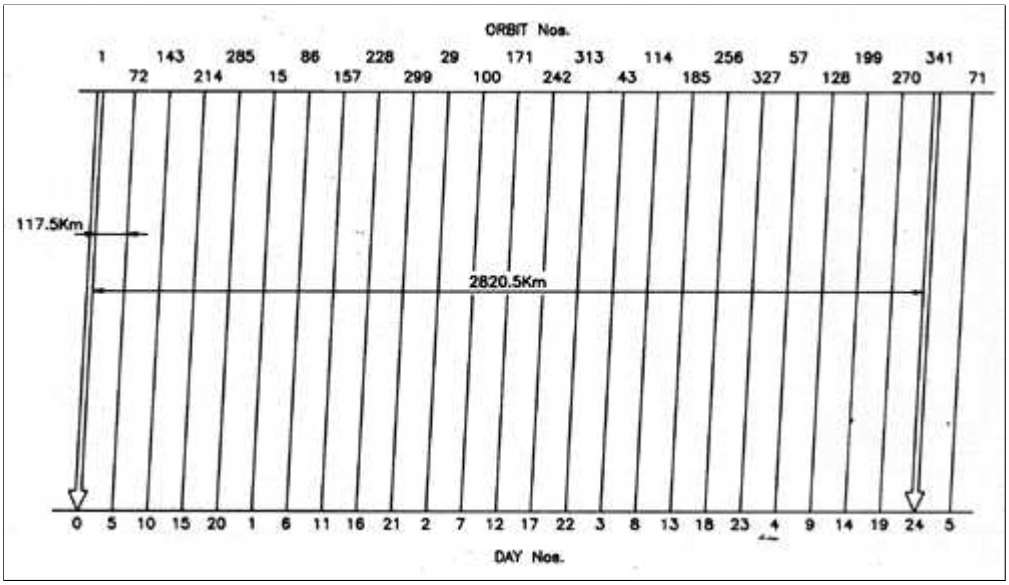


Figura 5.2: Patrón de órbitas.

5.3 Imágenes

Hasta ahora, hemos hablado del par satélite/sensor que proporciona las imágenes, pero no de las imágenes que hemos utilizado. Las imágenes tienen las siguientes características por defecto:

- `Proyección:` SOM
- `Elipsoide:` Internacional 1909
- `Tamaño del producto:` Escena completa (full scene)
- `Tipo:` Orientado a órbita
- `Corrección:` Sistemática
- `Método de re-muestreo:` Convolución cúbica

- `Tamaño típico`: 200-250 MB

Naturalmente, se pueden adquirir imágenes con otro tipo de procesamiento, pero su precio se dispara[4]. Recordar que no todas las imágenes son adecuadas para este tipo de aplicaciones, y que hay que considerar factores de vital importancia como son: condiciones atmosféricas, visibilidad, etc. Como norma general se suelen elegir imágenes con fecha de adquisición cercanas a la estación primaveral y veraniega, fundamentalmente los meses de Marzo y Septiembre, debido a las excelentes condiciones atmosféricas y visibilidad que presenta el término municipal de Málaga en esta época del año. Lo que nos proporciona un periodo de cobertura de 6 meses, suficiente para una aplicación de detección de cambios urbanos.

5.3.1 Cabecera

Tan importante como las imágenes, son los datos que acompañan a éstas. Datos geográficos, radiométricos, etc. que son imprescindibles para su gestión y aplicación. Las imágenes suministradas por Euromap GmbH se encuentran en un formato denominado *Fast Format Rev.C* que consta de una imagen RAW (en bruto) y un fichero de cabecera. El fichero de cabecera consta de tres registros ASCII de 1536 bytes, los registros *administrativo*, *radiométrico* y *geométrico* respectivamente.

1. El registro *administrativo* contiene información que identifica el producto, la escena y los datos necesarios para recuperar los datos del soporte magnético u óptico. Es decir, para poder importar los datos de la imagen, es necesario leer las entradas del registro administrativo.

2. El registro *radiométrico* contiene los coeficientes necesarios para convertir los valores digitales de la escena en valores de radiancia espectral.

3. El registro *geométrico* contiene información relativa la localización geodésica de la escena. Para alinear las imágenes con otras fuentes de datos (Geographic Information Systems), será necesario leer entradas del registro geométrico.

[4]El precio base de estas imágenes ronda los 2500 € aproximadamente.

A continuación mostramos un fichero de cabecera típico en el que se puede apreciar claramente los tres registros a los que hacíamos referencia anteriormente.

Administrativo ⇐

```
PRODUCT ID =99124080-01 LOCATION =012/04480B0       ACQUISITION DATE =20000206
SATELLITE =IRS 1D     SENSOR =PAN        SENSOR MODE =       LOOK ANGLE =  2.10
                        LOCATION =                  ACQUISITION DATE =
SATELLITE =           SENSOR =           SENSOR MODE =       LOOK ANGLE =
                        LOCATION =                  ACQUISITION DATE =
SATELLITE =           SENSOR =           SENSOR MODE =       LOOK ANGLE =
                        LOCATION =                  ACQUISITION DATE =
SATELLITE =           SENSOR =           SENSOR MODE =       LOOK ANGLE =
PRODUCT TYPE =ORBIT ORIENTED     PRODUCT SIZE =FULL SCENE
TYPE OF PROCESSING =SYSTEMATIC  RESAMPLING =CC
VOLUME #/# IN SET =01/01 PIXELS PER LINE =15043 LINES PER BAND =14160/14160
START LINE # =    1 BLOCKING FACTOR = 1 RECORD LENGTH =15043 PIXEL SIZE =  5.00
OUTPUT BITS PER PIXEL = 8 ACQUIRED BITS PER PIXEL = 6
BANDS PRESENT =P                              PRODUCT CODE =TRSCB02AZ
VERSION NO =IRS1DDPSV2R0        ACQUISITION TIME =11:31:04:597
```

Radiométrico ⇐

```
REV            CBIASES AND GAINS IN THE BAND ORDER AS ON THIS TAPE
       0.000000000000000        9.720000000000001
       0.000000000000000        0.000000000000000
       0.000000000000000        0.000000000000000
       0.000000000000000        0.000000000000000
       0.000000000000000        0.000000000000000
       0.000000000000000        0.000000000000000
       0.000000000000000        0.000000000000000
       0.000000000000000        0.000000000000000

SENSOR GAIN STATE =   4   4   4
```

Geométrico ⇐

```
GEOMETRIC DATA MAP PROJECTION =SOM  ELLIPSOID =INTERNATL_1909     DATUM =
USGS PROJECTION PARAMETERS =  6378388.000000000000000  6356911.946000000500000
       0.000000000000000       13.816406250000000        0.000000000000000
       0.000000000000000        0.000000000000000        0.000000000000000
     178.873990700000010        0.000000000000000      -13.794759299999981
       0.000000000000000        0.000000000000000        0.000000000000000
       0.000000000000000
UL = 0044544.2505W 370226.2612N  15971140.385    531287.558
UR = 0035556.5106W 365451.9790N  15975005.298    606402.611
LR = 0040454.2256W 361716.6895N  16045712.879    602828.075
LL = 0045418.6984W 362447.2513N  16041847.972    527713.056
CENTER = 0042225.8861W 363929.6201N  16008586.766    571275.313  7521  7080
OFFSET = -3071 ORIENTATION ANGLE =-13.82
SUN ELEVATION ANGLE =72.4 SUN AZIMUTH ANGLE =143.1
```

Registro administrativo

El registro administrativo contiene información que identifica el producto, la escena y los datos necesarios para importar los datos de la imagen. Estos son algunos de los campos de este registro:

`LOCATION (Bytes 13-23)`
Localización de la primera escena (ppp/rrrffss, donde ppp=path, rrr=row, ff=shift y ss=subescena o número de cuadrante).

`ACQUISITION DATE (Bytes 71-78)`
Fecha de adquisición de la primera escena (yyyyddmm, donde yyyy=año, dd=día y mm=mes.)

`SENSOR (Bytes 71-78)`
Sensor de la primera escena.

`PRODUCT TYPE (Bytes 655-672)`
Tipo del producto.

`TYPE OF PROCESSING (Bytes 741-821)`
Tipo de procesamiento.

`RESAMPLING (Bytes 765-766)`
Método de re-muestreo.

`PIXELS PER LINE (Bytes 843-847)`
Número de píxeles por línea (ancho de la imagen).

`LINES PER BAND (Bytes 865-869)`
Número de líneas por banda (alto de la imagen).

`PIXEL SIZE (Bytes 954-959)`
Tamaño del píxel (metros) – Resolución espacial.

`OUTPUT BITS PER PIXEL (Bytes 984-985)`
Bits de salida por píxel – Resolución espectral (imagen).

`ACQUIRED BITS PER PIXEL (Bytes 1012-1013)`
Bits de entrada por píxel (satélite) – Resolución espectral (sensor).

Registro radiométrico

El registro radiométrico contiene los coeficientes necesarios para convertir los valores digitales de la escena en valores de radiancia espectral. Estos son algunos de los campos de este registro:

`BIASES AND GAINS (Bytes 81-104 y 106-129)`
Valores de radiancia mínimo y máximo para la primera banda.

> **NOTA:** La ecuación que permite convertir valores digitales en valores de radiancia espectral (sensor) es la siguiente:
>
> $$L_{sensor,k} = \frac{DN}{DN_{max}}(L_{max,k} - L_{min,k}) + L_{min,k} \qquad (5.1)$$
>
> donde DN=Nivel de gris, $L_{sensor,k}$=Radiancia en el sensor para la banda k, DN_{max}=Nível de gris máximo (255), $L_{max,k}$=Radiancia máxima para la banda k (GAIN) y $L_{min,k}$=Radiancia mínima para la banda k (BIAS).

Registro geométrico

El registro geométrico contiene información relativa la localización geodésica de la escena. Estos son algunos de los campos de este registro:

`GEOMETRIC DATA MAP PROJECTION (Bytes 32-35)`
Proyección.
`ELLIPSOID (Bytes 48-65)`
Elipsoide.
`USGS PROJECTION PARAMETERS (Bytes 110-504 - Varios)`
Parámetros USGS[5] (U.S. Geological Survey) de proyección.

> **NOTA:** Estos parámetros son necesarios para cambiar el sistema de proyección, elipsoide, etc. La librería GCTP (General

[5] `http://www.usgs.gov`

Cartographic Transformation Package) suministrada por el USGS facilita este tipo de procesamiento.

`UL (Bytes 580-619 - Varios)`
Coordenada geodésica[6] de la esquina superior-izquierda de la imagen.

`UR (Bytes 646-699 - Varios)`
Coordenada geodésica de la esquina superior-derecha de la imagen.

`LR (Bytes 726-779 - Varios)`
Coordenada geodésica de la esquina inferior-derecha de la imagen.

`LL (Bytes 806-879 - Varios)`
Coordenada geodésica de la esquina inferior-izquierda de la imagen.

`ORIENTATION ANGLE (Bytes 995-1000)`
Ángulo de orientación de la escena (grados).

NOTA: Las ecuaciones que permiten obtener el Easting y el Northing para cualquier pixel de la imagen, a partir de las coordenadas de las esquinas son las siguientes:

$$\begin{aligned} P_{Easting} = [&(NP-P)(NL-L)UL_{Easting} + \\ &(P-1)(NL-L)UR_{Easting} + \\ &(NP-P)(L-1)LL_{Easting} + \\ &(P-1)(L-1)LR_{Easting}]/[(NP-1)(NL-1)] \end{aligned} \tag{5.2}$$

$$\begin{aligned} P_{Northing} = [&(NP-P)(NL-L)UL_{Northing} + \\ &(P-1)(NL-L)UR_{Northing} + \\ &(NP-P)(L-1)LL_{Northing} + \\ &(P-1)(L-1)LR_{Northing}]/[(NP-1)(NL-1)] \end{aligned} \tag{5.3}$$

donde $P_{Easting}$ y $P_{Northing}$ son el easting y northing del píxel, P y L la columna y fila del píxel, NP y NL el número de píxeles por línea (ancho) y número de líneas por banda (alto), $UL_{Easting}$ y $UL_{Northing}$ el easting y northing de la esquina superior-izquierda,

[6] Latitud/Longitud (grados, minutos y segundos) y Easting/Northing (metros).

$UR_{Easting}$ y $UR_{Northing}$ el easting y northing de la esquina superior-derecha, $LR_{Easting}$ y $LR_{Northing}$ el easting y northing de la esquina inferior-derecha y $LL_{Easting}$ y $LL_{Northing}$ el easting y northing de la esquina inferior-izquierda.

Capítulo 6

Procesamiento

6.1 Introducción

Según lo expuesto en el capítulo 4, todo proceso de interpretación y análisis viene precedido por dos etapas previas que permiten extraer con facilidad información de interés de los datos utilizados:

1. Preprocesamiento de la imagen
   (a) Corrección geométrica
   (b) Corrección radiométrica
2. Transformación de la imagen (diferencia de imágenes)

Las etapas de preprocesamiento y transformación –procesamiento– son cruciales para cualquier proceso de análisis e interpretación: la primera corrige radiométrica y geométricamente los datos de entrada, condición imprescindible para el estudio de series multi-temporales de imágenes; la segunda utiliza los datos de salida de la primera y genera otros que facilitan tanto la interpretación visual como la informática, reduciendo las regiones de estudio a posibles cambios ocurridos en la serie.

Como comentamos en la sección 1.2, los datos proporcionados por esta aplicación serán utilizados en el GIS de la gerencia de urbanismo del Iltmo. Ayuntamiento de Málaga, y por tanto será un analista humano el que, utilizando los datos obtenidos junto con los recursos geográficos de que dispone este GIS, determine la naturaleza de los cambios detectados. Será en esta

etapa, fuera del ámbito de este proyecto, en la que se determinará la legalidad o ilegalidad de los cambios detectados, tomando las medidas disciplinarias estipuladas por la ordenanza municipal, si se demostrara que el cambio detectado en la etapa de transformación es de origen humano y carece de la correspondiente licencia municipal.

6.2 Corrección geométrica

Para abordar la detección digital de cambios es preciso que las imágenes se ajusten con gran precisión, ya que de otro modo estaríamos detectando como cambios lo que sería sólo consecuencia de una falta de ajuste entre imágenes. El efecto puede llegar a ser muy grave, especialmente cuando se trata de detectar cambios en regiones con una gran variabilidad espacial, como es el caso de las urbanas, dependiendo también de la resolución espacial del sensor.

6.2.1 Antecedentes

Las correcciones geométricas de la imagen incluyen cualquier cambio en la posición que ocupan los píxeles que la forman. Por contraposición con las correcciones radiométricas, aquí en principio, no se pretende modificar sus niveles de gris[1], sino sólo su posición (sus coordenadas). Esta transformación puede basarse en funciones numéricas, que permiten modificar la geométrica de la imagen. La expresión general de este tipo de funciones sería la siguiente:

$$f(fila_{corregida}) = f_{fila}(fila, columna) \quad (6.1)$$

$$f(columna_{corregida}) = f_{columna}(fila, columna) \quad (6.2)$$

donde $fila_{corregida}$ y $columna_{corregida}$ son la fila y columna de la imagen corregida y $fila$ y $columna$ la fila y columna de la imagen de entrada (corrección geométrica *relativa* o *image-to-image*).

Si en lugar de utilizar la fila y columna de la imagen de entrada, utilizáramos las coordenadas de un mapa, la expresión general de este tipo funciones

[1] Teóricamente, los valores digitales de los píxeles no se modifican, pero como veremos a continuación, el trasvase de valores de la imagen original a la corregida, obliga en la mayoría de los casos, a modificarlos.

sería la siguiente:

$$f(fila_{corregida}) = f_{fila}(x, y) \quad (6.3)$$

$$f(columna_{corregida}) = f_{columna}(x, y) \quad (6.4)$$

donde x e y son las coordenadas del mapa (corrección geométrica *absoluta* o *image-to-map*).

Por lo tanto, esta transformación puede utilizarse tanto para georeferenciar una imagen, como para superponer dos o más imágenes entre sí. En el primer caso, se pretende encontrar una relación que transfiera los valores digitales de la imagen a su posición cartográfica, en la proyección requerida. En el segundo caso, las transformación geométrica de la imagen tiene por objetivo ajustarla a otra imagen que se considera como referencia, normalmente con objeto de realizar estudios multi-temporales.

La corrección geométrica puede abordarse de acuerdo a tres procedimientos [4]:

1. Corrección a partir de modelos orbitales

La corrección orbital pretende modelar las fuentes de error geométrico conocidas, aplicando transformaciones inversas a las que realiza el sensor en momento de la adquisición. Para ello es preciso disponer, con bastante precisión, de los parámetros orbitales de la plataforma y de las especificaciones del sensor (ver tablas 5.1 y 5.3). Gracias a ellas, pueden corregirse errores sistemáticos, como son los derivados de la rotación o curvatura terrestre y de la inclinación de la órbita. Habitualmente, al aplicar correcciones orbitales se utiliza el término "navegación" de la imagen, ya que se pretende localizar sobre una malla geográfica cada uno de los píxeles que la forman [12].

> **NOTA:** Como expusimos en el capítulo anterior, el tipo de procesamiento de las imágenes utilizadas en este proyecto, coincide con el obtenido con la corrección basada en modelos orbitales (`TYPE OF PROCESSING =SYSTEMATIC`).

2. Corrección a partir de puntos de control terrestres (GCPs)

La corrección a partir de GCPs trata de modelar el error geométrico de la imagen a partir de una serie de puntos con coordenadas conocidas, denominados puntos de control terrestre. En este caso, el error se modela inductivamente, ya que en las funciones de transformación se incluyen simultáneamente todas las fuentes de error, asumiendo, como es lógico, que esos puntos sean suficientemente representativos de la deformación geométrica que tiene la imagen.

Los dos métodos anteriores tienen sus ventajas e inconvenientes. El primer método favorece el procesamiento automático, puesto que los parámetros orbitales se reciben directamente con las imágenes y apenas se requiere intervención humana. Resulta, muy adecuado cuando el sensor no proporciona información fiable para localizar los puntos de control terrestres. Este es el procedimiento utilizado habitualmente para corregir imágenes de baja resolución. Su principal inconveniente es la escasa precisión cuando la telemetría del satélite no es muy exacta o cuando la imagen incluye errores no sistemáticos (aleatorios). Por otra parte, el segundo método resulta bastante tedioso, puesto que requiere una notable intervención humana para localizar adecuadamente los puntos de control, sin embargo ofrece una precisión muy alta cuando se trabaja sobre regiones e imágenes en donde es fácil la identificación de rasgos comunes.

3. Corrección a partir de modelos digitales de elevación (DEMs)

Las corrección geométrica basada en puntos de control es el procedimiento más utilizado en imágenes espaciales de alta resolución, y el que está disponible en la mayor parte de los programas comerciales. Los resultados son, en la mayor parte de los casos, suficientemente buenos. Sin embargo, en el caso de trabajar sobre regiones con alta variabilidad espacial, o con imágenes aéreas, en donde las deformaciones geométricas están estrechamente ligadas al modelado del relieve, sería necesario acudir a información topográfica. Para realizar una corrección geométrica del efecto topográfico es preciso contar, previamente, con modelos digitales de elevación ó DEMs (Digital Elevation

Model). Un modelo digital de elevación es una matriz numérica, con un formato muy similar al de una imagen, donde se almacena un valor de altitud, en lugar de un valor de radiancia. Es decir, el valor digital que define cada píxel en un DEM corresponde a su altitud sobre el nivel del mar.

A continuación describimos con detalle el procedimiento seguido en este proyecto.

Corrección a partir de puntos de control terrestres (GCPs)

Descartados los procedimientos de corrección geométrica basados en modelos (*modelos orbitales* y *modelos digitales del terreno*), debido a la escasa precisión de los parámetros orbitales disponibles y a las características topográficas del aérea de interés, centramos nuestro estudio en la corrección a partir de puntos de control terrestres (GCPs), tratando de solventar el principal inconveniente de este método, la importante intervención humana que requiere.

La corrección geométrica a partir de GCPs, asume que no conoce las fuentes de error, pero que pueden modelarse ajustando unas ecuaciones a un conjunto de puntos, de los que se conoce tanto las coordenadas de la imagen a corregir como las del mapa (corrección *absoluta*) o imagen (corrección *relativa*) de referencia. En ambos casos la corrección se realiza en tres etapas:

1. Localización de los GCPs comunes a la imagen a corregir y al mapa o imagen de referencia

2. Cálculo de las funciones de transformación[2] entre coordenadas de la imagen a corregir y las del mapa o imagen de referencia

3. Transferencia de los valores digitales de los píxeles originales a su nueva localización, definida por las funciones de transformación

Localización de los GCPs

Como podemos observar la calidad del ajuste dependerá de la precisión con que se localicen esos puntos, y de cómo de representativos sean esos puntos de

[2]Polinomios de ajuste.

los errores geométricos de la imagen. Una localización inexacta de los puntos, tanto en la imagen a corregir como en el mapa o imagen de referencia, o una distribución poco uniforme, provocará el cálculo de unas funciones de transformación incorrectas y, por lo tanto, una corrección geométrica errónea. En resumen, el establecimiento de GCPs constituye la etapa crucial del proceso de corrección, y la que demanda mayor dedicación humana.

Para que el ajuste entre imagen a corregir y mapa o imagen de referencia sea correcto, debemos considerar los siguientes aspectos en la etapa de selección de puntos de control:

1. El número
2. La localización
3. La distribución

El *número de puntos* depende del tamaño y de la complejidad geométrica de la imagen a corregir. En términos generales, cabe decir que cuanto mayor sea el grado del polinomio de ajuste, mayor será el número de puntos de control que se precisan. Matemáticamente sólo son necesarios 3 GCPs para un polinomio de ajuste de primer grado, 6 para uno de segundo grado y 10 para uno de tercer grado. Sin embargo, conviene superar con creces este número mínimo para garantizar un buen ajuste entre la imagen a corregir y el mapa o imagen de referencia. Para regiones pequeñas y/o con baja variabilidad espacial basta con seleccionar 10 ó 12 GCPs y utilizar un polinomio de ajuste de primer grado (lineal). En caso de regiones grandes y/o con alta variabilidad espacial, conviene seleccionar un número de GCPs mayor y utilizar polinomios de ajuste más complejos (segundo o tercer grado).

En cuanto a la *localización*, se recomienda que sean puntos claramente identificables en la imagen a corregir y en el mapa o imagen de referencia, preferiblemente aspectos humanos del paisaje no sujetos a cambios temporales frecuentes: cruces de carreteras o vías de ferrocarril, presas, etc. No conviene señalar puntos en la línea costera, pues el efecto de la marea puede modificar su localización geográfica.

La localización geográfica del término municipal de Málaga dificulta notablemente la selección automática de GCPs. Esto obliga a implementar

técnicas específicas para minimizar tanto como sea posible, sus efectos negativos. Por ejemplo, GCPs debidos a la marea, aleatorios (por ejemplo, barcos), etc. no contarán con sus correspondientes GCPs en el mapa o imagen de referencia.

Respecto a su *distribución*, conviene que estos puntos estén distribuidos uniformemente sobre toda la imagen. Esto evitará errores debidos a una ponderación excesiva de una región específica, en detrimento del resto de las regiones de la imagen. Esto es especialmente importante cuando la zona presente una alta variabilidad espacial y se utilicen polinomios de ajuste de segundo o tercer grado.

Localización automática de GCPs

La localización automática de GCPs es, en la mayoría de los casos, un proceso de dos etapas. La primera etapa consiste en identificar puntos o estructuras significativas en la imagen. En la segunda etapa, se intenta encontrar una correspondencia entre las estructuras extraídas de las imágenes, etapa que recibe el nombre de emparejamiento de imágenes (*image matching*). Existen dos aproximaciones principales para resolver este problema:

1. Métodos basados en correlación

2. Métodos simbólicos

Los métodos *basados en correlación* utilizan centros de ventanas como puntos de control. La localización de ventanas puede realizarse por correlación clásica [15], correlación entre bordes [21] o entre vectores [3]. Los cálculos de la correlación también pueden realizarse en el dominio de Fourier [1], mediante la utilización de representaciones jerarquizadas de la imagen [22] o por cálculo secuencial de disimilitud [2].

Los métodos *simbólicos* no trabajan directamente con los valores digitales de la imagen. En su lugar utilizan características de la escena tales como intersecciones entre líneas, bordes [19] o por ejemplo, los centros de gravedad de las regiones [7].

Sin embargo, los métodos que utilizan el cálculo de correlaciones no son apropiados para problemas en los que intervengan errores geométricos pro-

nunciados, puesto que el cálculo de la correlación normalizada es muy sensible a variaciones en la forma [10].

Los métodos basados en información simbólica extraída de la imagen, como pueden ser líneas, bordes, regiones homogéneas, etc. suelen soportar errores en la imagen de mayor importancia que las técnicas anteriores, sin embargo dependen del contenido de la imagen. Aún así, su robustez los hace ideales para nuestro propósito.

> **NOTA:** Independientemente del método utilizado, el principal escollo con el que nos encontramos a la hora de aplicar una técnica de selección automática de puntos, ya sea basada en correlación o en información simbólica, es la imposibilidad de asegurar un conjunto adecuado de puntos *uniformemente distribuidos*.

Cálculo de las funciones de transformación

Como comentamos anteriormente, la corrección geométrica de una imagen se realiza estableciendo unas funciones, que relacionan las coordenadas (fila,columna) de la imagen a corregir con las del mapa (abscisa,ordenada) o imagen (fila, columna) de referencia. De esta forma, puede estimarse que coordenadas de la imagen a corregir se corresponden con las coordenadas del mapa o imagen tomada como referencia. Las siguientes ecuaciones expresan de forma resumida la transformación a la que hacemos referencia:

$$\hat{x}_k = \sum_{i=0}^{m} \sum_{j=0}^{m-i} a_{i,j} x_k^i y_k^j \tag{6.5}$$

$$\hat{y}_k = \sum_{i=0}^{m} \sum_{j=0}^{m-i} b_{i,j} x_k^i y_k^j \tag{6.6}$$

donde $\hat{x}_k$ e $\hat{y}_k$ son las coordenadas estimadas de la imagen corregida, x_k e y_k las coordenadas del mapa o imagen de referencia, $a_{i,j}$ y $b_{i,j}$ los coeficientes de ajuste y m el grado del polinomio de ajuste.

Por ejemplo, si el grado del polinomio de ajuste y el número de puntos fuese 2 y n respectivamente, las ecuaciones quedarían:

1. $\hat{x}_k = \begin{cases} \hat{x}_1 &= a_0 + a_1x_1 + a_2y_1 + a_3x_1y_1 + a_4x_1^2 + a_5y_1^2 \\ \hat{x}_2 &= a_0 + a_1x_2 + a_2y_2 + a_3x_2y_2 + a_4x_2^2 + a_5y_2^2 \\ \cdots &= \cdots \\ \hat{x}_n &= a_0 + a_1x_n + a_2y_n + a_3x_ny_n + a_4x_n^2 + a_5y_n^2 \end{cases}$

2. $\hat{y}_k = \begin{cases} \hat{y}_1 &= b_0 + b_1x_1 + b_2y_1 + b_3x_1y_1 + b_4x_1^2 + b_5y_1^2 \\ \hat{y}_2 &= b_0 + b_1x_2 + b_2y_2 + b_3x_2y_2 + b_4x_2^2 + b_5y_2^2 \\ \cdots &= \cdots \\ \hat{y}_n &= b_0 + b_1x_n + b_2y_n + b_3x_ny_n + b_4x_n^2 + b_5y_n^2 \end{cases}$

Como puede comprobarse, se trata de una *regresión múltiple*, donde $a_0,\cdots,a_n$ son los coeficientes de regresión, $\hat{x}_1,\cdots,\hat{x}_n$ e $\hat{y}_1,\cdots,\hat{y}_n$ las variables dependientes y $x_1,\cdots,x_n$ e $y_1,\cdots,y_n$ las variables independientes.

Los coeficientes de las funciones de transformación se calculan a partir de las coordenadas de los GCPs seleccionados. El método habitual para obtenerlos consiste en aplicar un ajuste de *mínimo error cuadrático*, con operaciones de cálculo idénticas a las que requieren una regresión múltiple convencional.

El grado de ajuste obtenido se mide por la importancia de los *residuos*[3]. Cuanto mayor sea el residuo (para cada uno de los GCPs empleados en el proceso), menor será el ajuste entre variables dependientes e independientes. Extrapolando estos conceptos al proceso de corrección geométrica que nos ocupa, podemos medir la precisión del ajuste geométrico atendiendo al valor de los residuos de los GCPs seleccionados. El promedio de los residuos se conoce como *error cuadrático medio* ó RMS (Root Mean Squared). El error cuadrático medio se calcula como la raíz cuadrada de las desviaciones entre las coordenadas estimadas y las observadas.

$$RMS = \sqrt{\frac{\sum_{i=1}^{n}\left[(\hat{l}_i - l_i)^2 + (\hat{c}_i - c_i)^2\right]}{n}} \tag{6.7}$$

donde RMS es el error cuadrático medio, $\hat{l}_i$ y $\hat{c}_i$ las coordenadas estimadas del punto i, l_i y c_i las coordenadas observadas del punto i y n el número de puntos.

[3]Diferencia entre el valor estimado por la regresión y el observado (o real).

También puede calcularse el RMS para cada punto, como la raíz cuadrada de las desviaciones entre las coordenadas estimadas y las observadas para ese punto.

$$RMS_i = \sqrt{(\hat{l}_i - l_i)^2 + (\hat{c}_i - c_i)^2} \tag{6.8}$$

donde RMS_i=Error cuadrático del punto i.

Como podemos observar en la figura 6.1, la distancia entre las coordenadas observadas y las estimadas coincide con el RMS de un punto. En consecuencia, también podemos denominar al RMS de un punto como error longitudinal (EL).

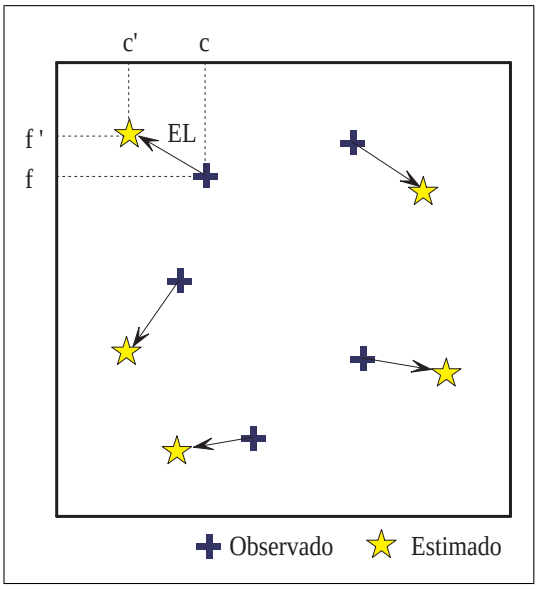


Figura 6.1: Error longitudinal (EL).

El promedio de los EL para todos los puntos se conoce como *error longitudinal medio* (ELM). El error longitudinal medio nos permite cuantificar la distancia media a la que se encuentran las coordenadas reales y las estimadas por la regresión.

$$ELM = \frac{\sum_{i=1}^{n} \sqrt{(\hat{l}_i - l_i)^2 + (\hat{c}_i - c_i)^2}}{n} \tag{6.9}$$

donde ELM es el error longitudinal medio.

Como comentamos anteriormente, tanto el RMS como el ELM pueden utilizarse para evaluar la calidad general de ajuste. Si el promedio supera un

cierto valor (umbral) indicado previamente (normalmente igual o inferior a un píxel), será conveniente corregir aquellos puntos con altos EL o eliminarlos si la posición es dudosa y, si es posible, elegir otros nuevos.

> **NOTA:** Tan importante es la selección automática de GCPs como su posterior validación. No todos los GCPs son igualmente válidos para el ajuste. Por lo tanto, es necesario desarrollar técnicas que permitan seleccionar y eliminar puntos de forma eficiente, tratando de lograr que el ajuste sea óptimo. Es decir, políticas que *seleccionen* GCPs que no satisfacen los criterios de calidad impuestos y los *eliminen*.

Transferencia de los valores digitales originales a su nueva localización

Las funciones de transformación permiten calcular la posición correcta de cada píxel, pero no producen una imagen nueva, puesto que sólo trasvasan coordenadas y no valores digitales. Es decir, con estas funciones de transformación puede crearse una nueva imagen, posicionada correctamente, pero vacía. El objetivo de la última fase de la corrección geométrica es, precisamente, el trasvase de los valores digitales originales a su nueva localización.

El problema resulta más complejo de lo que pudiera pensarse en un principio. Idealmente, cada píxel de la imagen corregida debería corresponderse con un sólo píxel en la imagen original. Lo normal es que no sea así, y que el píxel de la imagen corregida se corresponda con varios de la imagen original.

Como expusimos en la sección 4.4, el trasvase de valores digitales de la imagen original a su nueva localización en la imagen corregida puede realizarse por tres métodos:

- Vecino más próximo
- Interpolación bilinear
- Convolución cúbica

La elección entre uno de los tres métodos depende de la finalidad del proceso y de los recursos informáticos disponibles. Si se pretende corregir una

imagen clasificada, el método del *vecino más próximo* es la elección obligada, ya que es el único que preserva los valores digitales originales. Si, por el contrario, se pretende facilitar el análisis visual, habrá que optar por métodos de interpolación más elaborados (*interpolación bilinear* o *convolución cúbica*).

6.2.2 Solución aportada

Según lo expuesto hasta el momento, la *automatización* del proceso de corrección geométrica requiere consideraciones de diversa índole: la localización automática de GCPs representativos de la deformación geométrica que presenta la imagen, uniformemente distribuidos y en número suficiente; el cálculo de las funciones de transformación, incluido el desarrollo de políticas de *elección* y *eliminación* de GCPs de "baja calidad" y por supuesto, la transferencia de los valores digitales originales a su nueva localización. Estas son las soluciones aportadas para cada una de las etapas del proceso de corrección geométrica:

> **NOTA:** Como vimos en la sección 5.3, las imágenes utilizadas están correctamente georeferenciadas, por lo tanto, el proceso de corrección geométrica se reduce a corregir de forma relativa (*image-to-image*) las imágenes de la serie multi-temporal.

Localización de los GCPs

Como comentamos anteriormente la calidad del ajuste dependerá de la precisión con que se localicen los GCPs, y de cómo esos puntos sean suficientemente representativos de los errores geométricos de la imagen. Una localización inexacta de los puntos, tanto en la imagen a corregir como en la imagen de referencia, o una distribución poco uniforme, provocará el cálculo de unas funciones de transformación incorrectas y, por tanto, una inadecuada corrección geométrica. Nosotros pretendemos automatizar esta etapa y reducir, en la medida de lo posible, el grado de intervención humana.

Para que el ajuste entre imagen a corregir e imagen de referencia sea correcto, debemos considerar los siguientes aspectos en la etapa de selección de puntos de control:

1. El número
2. La localización
3. La distribución

Para asegurar el *número* y la *distribución uniforme* de los GCPs se ha optado por subdividir las imágenes de la serie multi-temporal en *tiles* (lozas), tomar una de ellas como referencia y realizar el ajuste del resto con respecto a ésta[4].

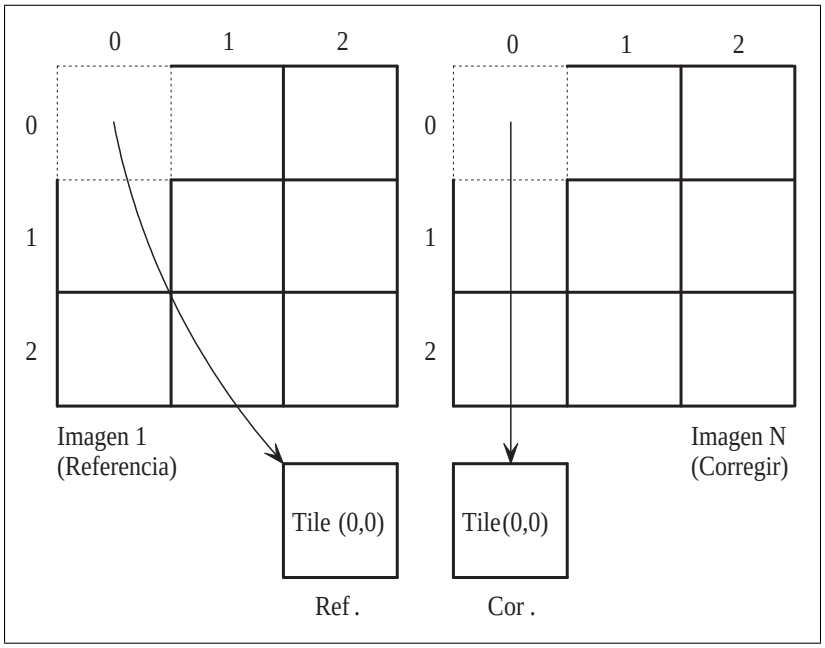


Figura 6.2: Subdivisión de las imágenes en *tiles*.

Una vez subdivididas, tanto la imagen a corregir como la de referencia, tomamos el primer *tile* de ambas imágenes, buscamos un punto (característica de interés) en el *tile* de la imagen de referencia, a continuación, tratamos de localizarlo en el *tile* de la imagen a corregir –búsqueda y seguimiento– (*tracking*). En caso de localizarlo, almacenamos las coordenadas y repetimos el proceso para los *tiles* restantes.

[4]El proceso de ajuste se realiza dos a dos (imagen de referencia e imagen a corregir).

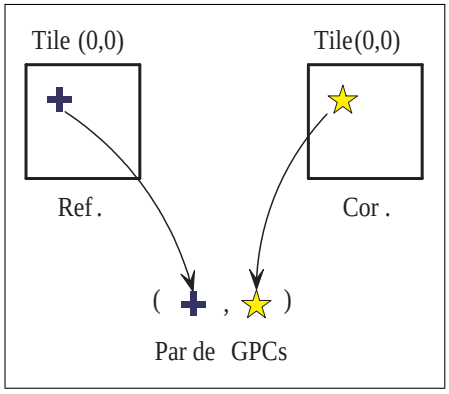


Figura 6.3: Búsqueda y seguimiento de puntos en pares de *tiles*.

El resultado de este proceso es un array de pares de puntos (o características de interés) uniformemente distribuidos, comunes a ambas imágenes. Naturalmente, el número de puntos depende directamente del tamaño del *tile* elegido y de las imágenes. Para un tamaño de imagen constante, el número de puntos es inversamente proporcional a la tamaño del *tile* elegido. Cuanto mayor sea el tamaño del *tile* menor será el número de puntos a localizar y viceversa.

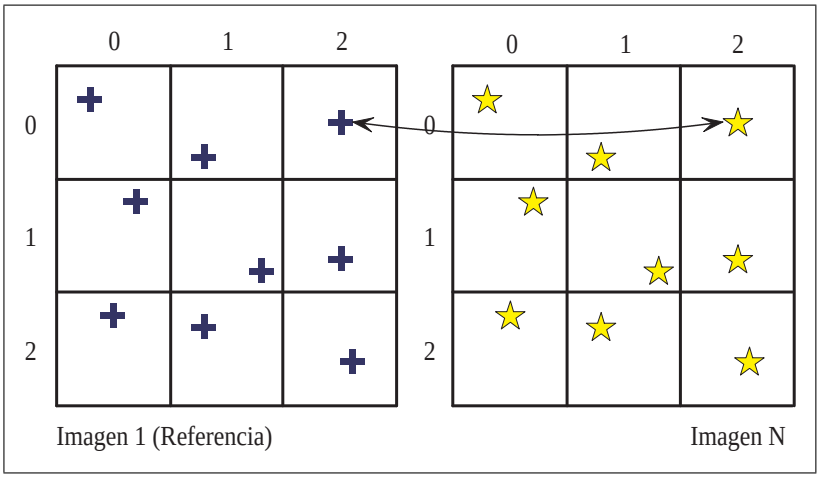


Figura 6.4: Distribución de los puntos en las imágenes.

Este proceso operativo obliga a utilizar imágenes con las mismas dimensiones (alto y ancho) y con un grado importante de solape (aproximadamente un 75%), ya que la localización de puntos se realiza *tile* a *tile* (*tracking* local),

y no sobre la totalidad de las imágenes (*tracking* global). Sin embargo, aseguramos en principio, el número de puntos deseado y su distribución uniforme.

Otra mejora sustancial, consiste en introducir una región de interés en el proceso de localización. Es decir, trazamos un polígono de interés (POI) que abarque aquella región de las imágenes a la que restringimos el proceso de búsqueda y seguimiento. Este concepto de polígono de interés es crucial, como veremos en capítulos posteriores. Los POIs restringen el proceso de corrección geométrica a una determinada región de interés, eliminando de un plumazo regiones de escaso interés urbanístico, que por sus condiciones topográficas pudieran afectar negativamente el proceso de corrección geométrica (el mar, las montañas, etc.).

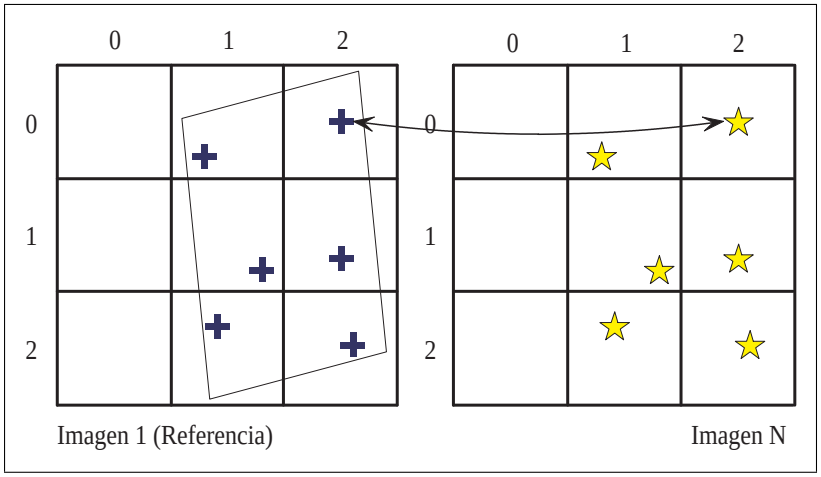


Figura 6.5: Polígono de interés.

Este sería el comportamiento ideal que cabría esperar para un correcto ajuste geométrico. Sin embargo, en la práctica esto no es así. La búsqueda y seguimiento de puntos no es del todo exacta y es posible que haya pares de *tiles* para los que no detecten puntos. Bien, porque no los localice en el *tile* de la imagen de referencia o en el *tile* de la imagen a corregir. Ante esta circunstancia, el único modo de solucionar este problema, a parte de la intervención humana, consiste en relajar las condiciones de búsqueda y seguimiento de puntos. Estos y otros aspectos serán tratados posteriormente, cuando describamos pormenorizadamente la técnica de búsqueda y seguimiento de puntos, verdadero "corazón" de la etapa de corrección geométrica.

Localización automática de GCPs

Hasta ahora hemos hablado continuamente de búsqueda y seguimiento de puntos, pero sin entrar en demasiada profundidad. El verdadero "corazón" de la etapa de corrección geométrica lo constituye una librería experimental de funciones, la *KLT Feature Tracker 1.1.5* desarrollada por Stan Birchfield, que implementan un motor de búsqueda y seguimiento (*tracker*) de características en *frames* contiguos (visión estéreo).

KLT Feature Tracker 1.1.5

El *tracker* de características está basado en el primer trabajo de Lucas y Kanade [13] y fue desarrollado completamente por Tomasi y Kanade [20], pero la única descripción publicada, fácilmente accesible está contenida en el artículo de Shi y Tomasi [18]. Recientemente, Tomasi propuso una ligera modificación que hace el computo simétrico con respecto a las dos imágenes. En este artículo se desarrolla teóricamente un "buscador" y "seguidor" óptimo de puntos de interés, que permite medir la calidad de las puntos detectados durante el *tracking*.

La verdadera potencia de este método, basado en información simbólica, radica en lo siguiente:

1. Detecta oclusiones (*occlusions*), exclusiones (*disocclusions*) y características sin correspondencia con elementos del mundo real (calles, ríos, etc).

2. Comportamiento muy robusto frente a *frames* contiguos que presentan deformaciones afines (escala, rotación, perspectiva, etc.).

En otras palabras, a) no depende del contenido de la imagen y b) detecta y sigue características en imágenes que presentan deformaciones afines. Las series de imágenes de satélite suelen presentar este tipo de deformaciones (escala, rotación, perspectiva, etc.). Esto lo hace especialmente útil para nuestro propósito, aún con las oportunas matizaciones. En nuestro caso tratamos de detectar y seguir puntos en *tiles* de imágenes separadas temporalmente 6 meses[5], y no unos milisegundos, como ocurriría en aplicaciones de visión estéreo,

[5]El grado de cambio de una imagen con respecto a otra puede ser realmente importante.

por lo tanto, el proceso de adaptación es crucial para obtener los resultados adecuados.

Shi y Tomasi proporcionan un método basado en dos modelos de movimiento: el primero basado en *traslaciones* y el segundo basado en *cambios afines*. El modelo basado en traslaciones es especialmente fiable cuando la translación de la cámara entre imágenes es pequeña, en caso contrario, es preciso considerar los cambios afines entre ellas. Cuando la cámara se mueve, los patrones de intensidad de la imagen cambian de un modo complejo. Sin embargo, estos cambios (deformación de los bordes, etc.) pueden ser descritos en términos de *movimiento* de los píxeles de la imagen:

$$I(x, y, t + \tau) = I(x - \xi(x, y, t.\tau), y - \eta(x, y, t, \tau)) \tag{6.10}$$

Así, una imagen posterior tomada en el instante $t + \tau$ puede ser obtenida moviendo cada punto de la imagen actual, tomada en el instante t, una cantidad adecuada. La cantidad de movimiento $\delta = (\xi, \eta)$ se llama *desplazamiento* del punto $\mathbf{x} = (x, y)$.

El vector de desplazamiento δ es una función de la posición $\mathbf{x}$ de la imagen. Variaciones en δ son a menudo apreciables incluso dentro de las pequeñas ventanas utilizadas en el *tracking*. Tiene entonces más sentido hablar del movimiento de una ventana de características, ya que hay diferentes desplazamientos dentro de la misma ventana. Un *campo de movimiento afín* es una mejor representación:

$$\delta = \quad D\mathbf{x} + \mathbf{d} \tag{6.11}$$

$$D = \begin{bmatrix} d_{xx} & d_{xy} \\ d_{yx} & d_{yy} \end{bmatrix}$$

donde D es la matriz de deformación y d la translación del centro de la ventana de características.

Las coordenadas $\mathbf{x}$ de la imagen son medidas con respecto al centro de la ventana. Entonces, un punto $\mathbf{x}$ de la primera imagen I se traslada al punto $A\mathbf{x} + \mathbf{d}$ en la segunda imagen J, donde $A = \mathbf{1} + D$ y $\mathbf{1}$ es la matriz identidad de 2×2.

$$J(A\mathbf{x} + \mathbf{d}) = I(\mathbf{x}) \tag{6.12}$$

Dadas dos imágenes I y J y una ventana de la imagen I, realizar el *tracking* significa determinar los seis parámetros que aparecen en la matriz de deformación D y en el vector de desplazamiento d. La calidad de esta estimación depende del tamaño de la ventana de características, la textura de la imagen dentro de ella, y del movimiento de la cámara entre imágenes. Cuando el tamaño de la ventana es pequeño, la matriz D es más difícil de estimar, porque las variaciones del movimiento dentro de la ventana son más pequeñas y por lo tanto menos fiables. Sin embargo, las ventanas pequeñas son preferibles para el *tracking* ya que son menos sensibles a discontinuidades importantes. Por esta razón, es preferible un modelo de *translación puro* durante el *tracking*, donde la matriz de deformación D es cero:

$$\delta = \mathbf{d} \tag{6.13}$$

La mejor combinación de estos dos modelos de movimiento es a) *translación pura* durante el *tracking*, debido su alta fiabilidad y exactitud con traslaciones pequeñas entre imágenes y b) *movimiento afín* para comparar características entre imágenes y monitorizar su calidad.

Cálculo de las funciones de transformación

Una vez descrita la técnica empleada para detectar y seguir puntos de interés en pares de *tiles*, pasamos a detallar el tipo de funciones de transformación empleadas en la etapa de corrección geométrica. Como comentamos anteriormente, el cálculo de las funciones de transformación se reduce al cálculo de una *regresión múltiple* empleando un ajuste de *mínimo error cuadrático*.

Los polinomios de ajuste utilizados son de la forma:

1. $\hat{x}_k = \begin{cases} \hat{x}_1 &= a_0 + a_1x_1 + a_2y_1 + a_3x_1y_1 \\ \hat{x}_2 &= a_0 + a_1x_2 + a_2y_2 + a_3x_2y_2 \\ \cdots &= \cdots \\ \hat{x}_n &= a_0 + a_1x_n + a_2y_n + a_3x_ny_n \end{cases}$

2. $\hat{y}_k = \begin{cases} \hat{y}_1 &= b_0 + b_1x_1 + b_2y_1 + b_3x_1y_1 \\ \hat{y}_2 &= b_0 + b_1x_2 + b_2y_2 + b_3x_2y_2 \\ \cdots &= \cdots \\ \hat{y}_n &= b_0 + b_1x_n + b_2y_n + b_3x_ny_n \end{cases}$

Estos son algunos ejemplos de transformación aplicables a una imagen digital con estos polinomios de ajuste:

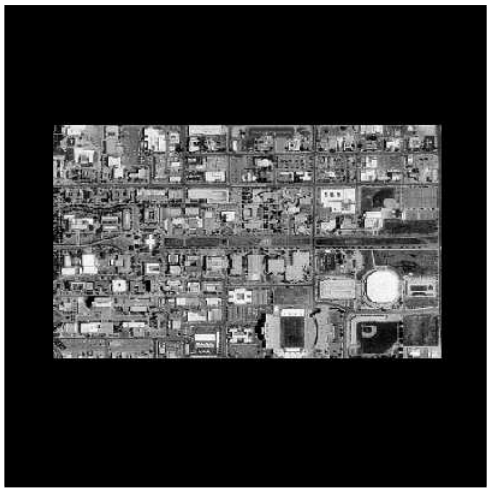


Figura 6.6: Imagen original.

Puntos de control: 3

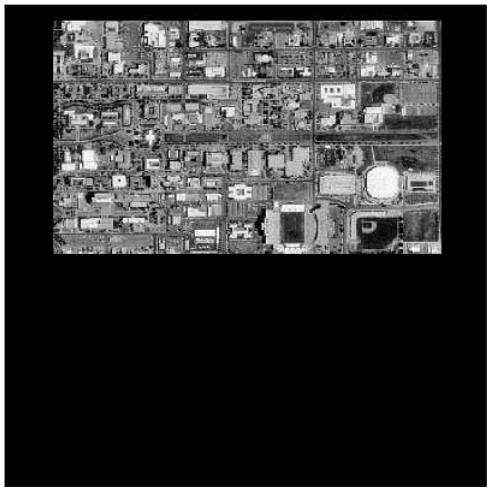
GCP	x	y	x'	y'
1	100	300	150	300
2	300	300	350	300
3	250	350	300	350

Número de términos: 3

Funciones de transformación:

$$\begin{aligned} x = & \quad -50 + x' \\ y = & \quad y' \end{aligned}$$

Figura 6.7: Desplazamiento en x.



Puntos de control: 3

GCP	x	y	x'	y'
1	100	300	100	200
2	300	300	300	200
3	250	350	250	250

Número de términos: 3

Funciones de transformación:

$$\begin{aligned} x &= x' \\ y &= 100 + y' \end{aligned}$$

Figura 6.8: Desplazamiento en y.

Puntos de control: 3

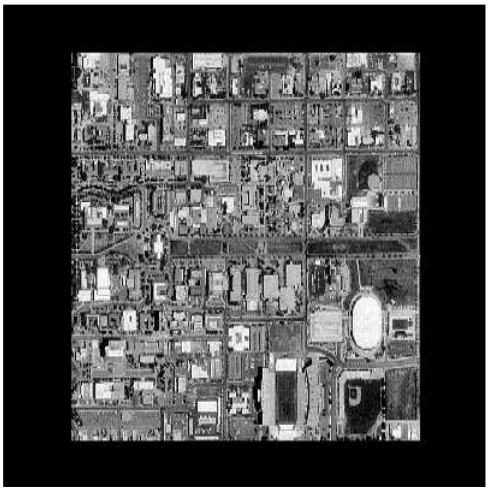
GCP	x	y	x'	y'
1	150	300	125	300
2	150	200	125	200
3	350	200	375	200

Número de términos: 3

Funciones de transformación:

$$\begin{aligned} x &= 50 + 0.8x' \\ y &= y' \end{aligned}$$

Figura 6.9: Escalado en x.



Puntos de control: 3

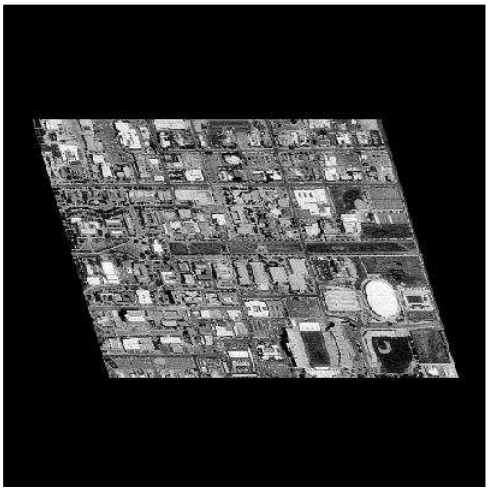
GCP	x	y	x'	y'
1	100	300	100	325
2	300	300	300	325
3	250	200	250	175

Número de términos: 3

Funciones de transformación:

$$\begin{aligned} x &= x' \\ y &= 83.3 + 0.667y' \end{aligned}$$

Figura 6.10: Escalado en y.



Puntos de control: 4

GCP	x	y	x'	y'
1	180	150	150	150
2	320	150	290	150
3	180	350	210	350
4	320	350	350	350

Número de términos: 4

Funciones de transformación:

$$\begin{aligned} x &= 75.0 + x' - 0.3y' \\ y &= y' \end{aligned}$$

Figura 6.11: Perspectiva en x.

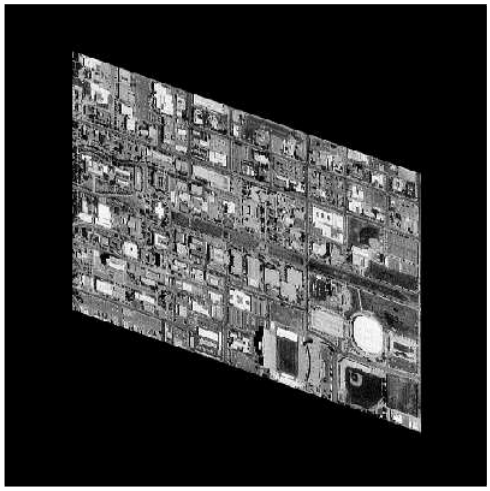


Figura 6.12: Perspectiva en y.

Puntos de control: 4

GCP	x	y	x'	y'
1	180	150	180	120
2	180	320	180	290
3	350	150	350	180
4	350	320	350	350

Número de términos: 4

Funciones de transformación:

$$x = x'$$
$$y = 93.5 - 0.353x' + y'$$

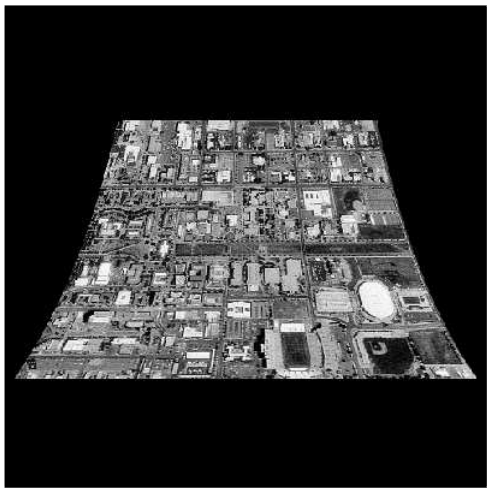


Figura 6.13: Escalado en x/y dependiente.

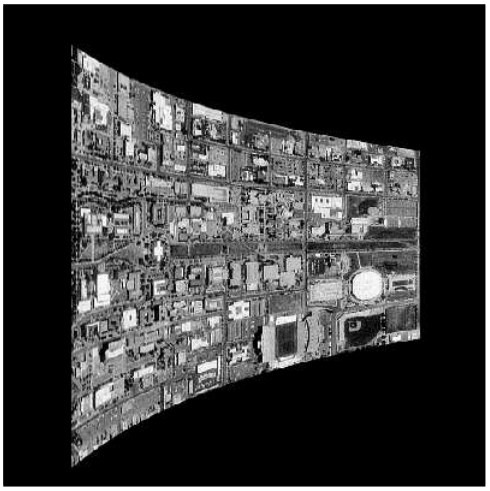
Puntos de control: 4

GCP	x	y	x'	y'
1	180	150	195	150
2	320	150	305	150
3	180	350	165	350
4	320	350	335	350

Número de términos: 4

Funciones de transformación:

$$x = -152 + 161x' + 0.5y' - 0.002x'y'$$
$$y = y'$$



Puntos de control: 4

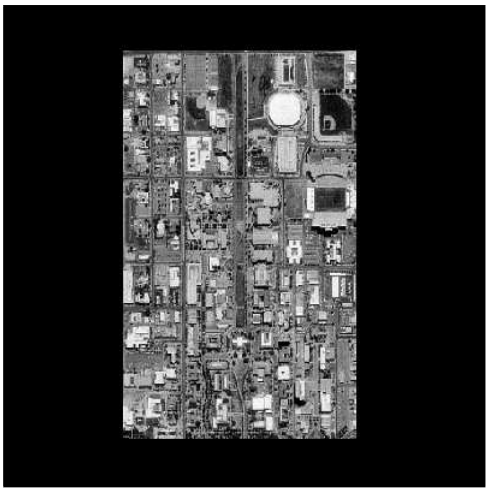
GCP	x	y	x'	y'
1	180	150	180	135
2	180	320	180	335
3	350	150	350	165
4	350	320	350	305

Número de términos: 4

Funciones de transformación:

$$x = x'$$
$$y = 126 - 0.5x' + 0.4y' + 0.002x'y'$$

Figura 6.14: Escalado en y/x dependiente.



Puntos de control: 4

GCP	x	y	x'	y'
1	237	244	237	254
2	237	254	246	254
3	246	254	246	244
4	246	244	237	244

Número de términos: 4

Funciones de transformación:

$$x = 466 - 0.9y'$$
$$y = -19.3 + 1.1x'$$

Figura 6.15: Rotación.

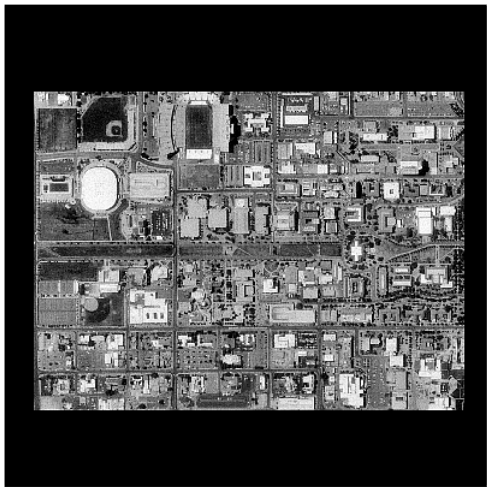


Figura 6.16: Inversión en el centro.

Puntos de control: 4

GCP	x	y	x'	y'
1	69	117	426	381
2	426	381	69	117
3	426	117	69	381
4	69	381	426	117

Número de términos: 4

Funciones de transformación:

$$x = 495 - x'$$
$$y = 498 - y'$$

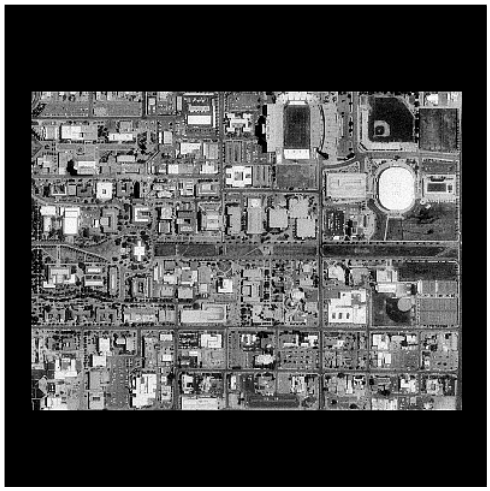


Figura 6.17: Inversión en x.

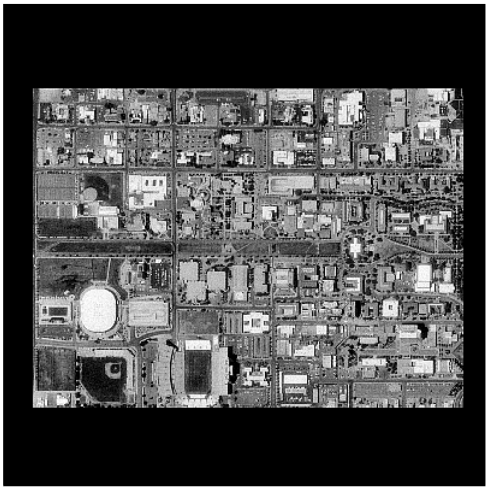
Puntos de control: 4

GCP	x	y	x'	y'
1	69	117	69	381
2	69	381	69	117
3	426	117	426	381
4	426	381	426	117

Número de términos: 4

Funciones de transformación:

$$x = x'$$
$$y = 498 - y'$$



Puntos de control: 4

GCP	x	y	x'	y'
1	69	117	426	117
2	69	381	426	381
3	426	117	69	117
4	426	381	69	381

Número de términos: 4

Funciones de transformación:

$$x = 495 - x'$$
$$y = y'$$

Figura 6.18: Inversión en y.

Como se puede comprobar en las figuras anteriores, el número de términos del polinomio elegido, permite corregir una gran variedad de deformaciones geométricas. Un número menor de términos limita las transformaciones geométricas consideradas y un número mayor puede introducir transformaciones indeseadas en la imagen (deformaciones) provocadas por una selección imprecisa de los GCPs. Por lo tanto, hay que alcanzar una solución de compromiso que nosotros hemos estimado en 4 términos (correcciones lineales y no lineales), suficientes para el tipo de procesamiento a realizar.

El paso siguiente es elegir una política adecuada para eliminar GCPs conflictivos. Como ya comentamos en su momento, se pueden emplear métodos matemáticos basados en residuos para medir la calidad del ajuste. El empleo del error medio longitudinal (ELM) puede enmascarar puntos con desviaciones importantes, si el número de puntos es alto, con objeto de minimizar este efecto, hemos optado por utilizar el error cuadrático medio (RMS) que presenta un comportamiento mucho más robusto [7].

La política de elección de candidatos considerada es simple y efectiva, y trata el minimizar el número de puntos elegidos, puesto que la eliminación de gran cantidad de puntos puede acabar con un ajuste global basado en información local (efecto del todo indeseado). Básicamente, la política de elección de puntos consiste en estudiar aquellos GCPs que presentan errores

longitudinales (EL) superiores a un pixel (5 metros), seleccionando para su posterior eliminación aquel que presente una mayor desviación. El punto se elimina si y sólo si el error cuadrático medio de todos los puntos supera el pixel. Este proceso se repite hasta que el error cuadrático medio sea igual o inferior a un pixel. La eliminación del punto que presenta una mayor desviación asegura una rápida convergencia.

Transferencia de los valores digitales originales a su nueva localización

Como comentamos anteriormente, la elección de uno de los tres métodos de trasvase depende de la finalidad del proceso y de los recursos informáticos disponibles. En nuestro caso, la calidad de la imagen final prima por encima de los recursos informáticos necesarios. Por lo tanto, atendiendo a las ventajas (y desventajas) de los métodos: vecino más próximo, interpolación bilinear y convolución cúbica, nos hemos decantado por este último, puesto que, aunque modifica los valores digitales de la imagen original, la imagen final se asemeja mucho más a la imagen inicial, sin líneas quebradas como ocurría en el método del vecino más próximo y con una calidad mayor que la proporcionada por el método de la interpolación bilinear.

Es importante recordar, la trascendencia del método de trasvase elegido, puesto que en un proceso de detección de cambios en regiones urbanas se requiere una precisión de subpíxel. La utilización del método del vecino más próximo, no modifica los niveles de gris, pero en su defecto, introduce fragmentación en la imagen. Esta fragmentación no pasa desapercibida en el proceso de detección, que evidentemente detecta como hipotéticos cambios de origen humano, cuando en realidad, se deben a una mala selección del método de trasvase.

6.2.3 Innovaciones

Básicamente, las innovaciones introducidas con esta técnica corrección geométrica son las siguientes:

1. Introducimos el concepto de *tiling* para asegurar un número conocido y uniformemente distribuido de GCPs.

2. Introducimos el concepto de polígono de interés (POI) para asegurar un ajuste óptimo en la región de estudio, evitando los posibles efectos negativos de considerar regiones poco fiables y de escaso interés urbanístico en el proceso de localización de GCPs (como el mar, regiones montañosas, etc.).

3. Empleamos una técnica de detección y seguimiento de puntos de interés –*tracking*– basada en información simbólica, independiente del contenido de las imágenes y capaz de seguir puntos de interés en imágenes que presentan deformaciones afines.

4. Empleamos una técnica de elección de GCPs candidatos para su eliminación, con objeto ajustar de forma óptima el polinomio y evitar desviaciones importantes.

En resumen, localizamos de forma automática GCPs distribuidos uniformemente y en número suficiente para aplicar correctamente un ajuste de mínimo error cuadrático. Y aunque no se podría considerar en sí mismo una innovación, utilizamos convolución cúbica para trasvasar los niveles digitales de la imagen original a la corregida, técnica que no se utiliza habitualmente, debido a la enorme cantidad de recursos que requiere, pero que presenta unos resultados excelentes.

Con esta técnica hemos tratado de reducir al máximo la intervención humana en la etapa del proceso de detección de cambios que más la requiere. Como veremos en capítulos posteriores, es posible que la intervención humana sea necesaria en etapas como las destinadas a elegir puntos candidatos para su eliminación, que requieren de cierta supervisión.

En las figuras 6.19, 6.20 y 6.21 podemos observar gráficamente la etapa de *corrección geométrica*. También podemos observar los GCPs detectados en la imagen de referencia y seguidos en la imagen a corregir, representados con una cruz (+), y los puntos eliminados en el proceso de ajuste, representados con un asterisco (∗).

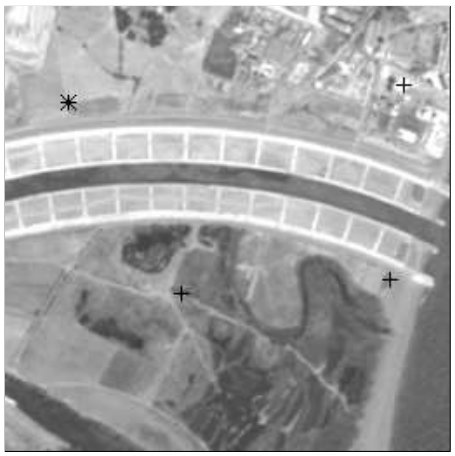


Figura 6.19: Imagen de referencia.

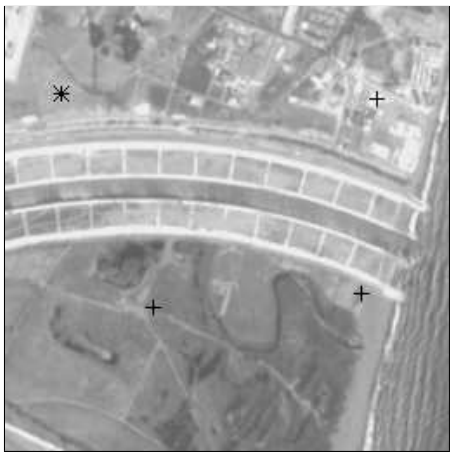


Figura 6.20: Imagen a corregir.

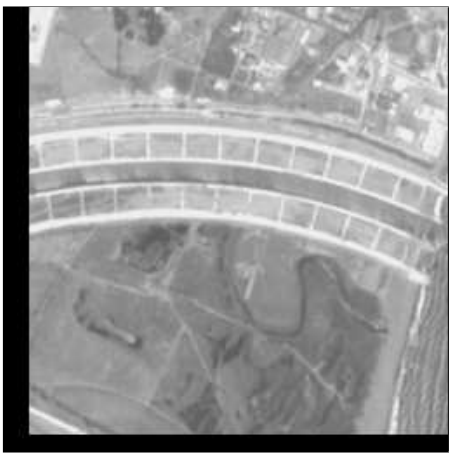


Figura 6.21: Imagen corregida geométricamente.

6.3 Corrección radiométrica

En el proceso de adquisición de imágenes de satélite y aéreas son numerosos los factores que afectan a la respuesta radiométrica, como las variables de iluminación, la absorción y dispersión atmosférica, la ganancia y el desfase del sensor, etc. Este problema se acusa especialmente en la realización de mosaicos digitales a partir de fotografías aéreas y en los estudios multi-temporales con imágenes de satélite, donde no sólo cambian las características de ilumi-

nación, transmisión y adquisición, sino que también existen modificaciones en los elementos del paisaje, debidas a causas tanto naturales como artificiales.

6.3.1 Antecedentes

En términos generales, el problema de la diferencia radiométrica entre dos fotografías se suele resolver mediante la corrección lineal de los histogramas [23], modificando el contraste y el brillo. Estas correcciones responden a transformaciones del tipo:

$$\hat{x}_k = a_k x_k + b_k \tag{6.14}$$

donde k es la banda espectral o componente de color, a el contraste, b el brillo y x y $\hat{x}$ el valor digital (nivel de gris) original y corregido respectivamente.

La normalización radiométrica puede abordarse de acuerdo a cuatro procedimientos:

- *Ajuste lineal* de los valores *máximo* y *mínimo* de la imagen que se va a normalizar con respecto a la de referencia.
- *Ajuste lineal* en función de la *media* y la *desviación típica*, mediante el cual se consigue que las características estadísticas de los valores de intensidad de la imagen corregida sean iguales a las de la imagen que se toma como referencia.
- *Regresión simple* global de las dos imágenes. Los coeficientes para la normalización se obtienen mediante un ajuste de error mínimo cuadrático.
- *Adaptativo bilineal* [16]. Se divide la imagen en *ventanas*, conectadas entre ellas por *puntos de malla*. Sobre cada uno de estos puntos se calcula la recta de regresión en función de la media y desviación típica de un determinado número de vecinos del punto. El ajuste final aplicado a cada píxel se calcula mediante interpolación bilineal de los cuatro puntos de malla que lo rodean.

Para la aplicación de los distintos métodos de normalización se pueden seleccionar los siguientes tipos de muestras:

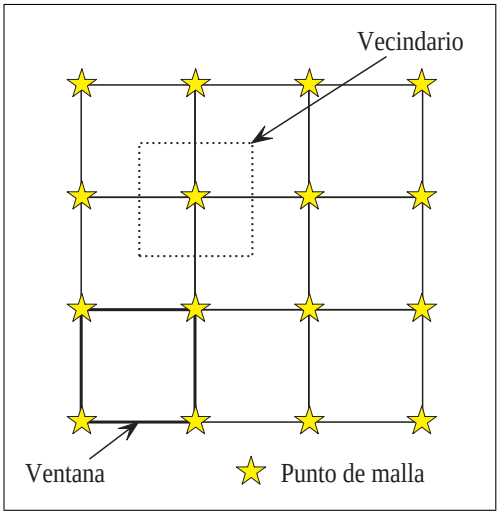


Figura 6.22: Esquema del método adaptativo bilineal.

- La imagen *completa*, formada por todos los píxeles que la componen.
- Las zonas *invariantes*, que son las denominadas sin-cambio [6], y que está formada por todos los píxeles interiores a una franja que envuelve a la recta de unión entre los centros de los grupos dominantes que aparecen en el diagrama de dispersión de las dos imágenes. De esta forma se seleccionan los píxeles que deben su variación radiométrica únicamente a las condiciones de toma y no a la variación temporal de la zona. Ésta puede considerarse la selección más objetiva.
- Las zonas *pseudo-invariantes*, que corresponden a aquéllas que, por criterios más subjetivos, se puede considerar que no han variado. Se suelen aplicar dos criterios, uno *específico*, consistente en la selección de zonas muy localizadas y con una similitud radiométrica en su entorno, y otro, más general o *global*, por el que se seleccionan zonas más amplias que globalmente puede considerarse que no han variado.

La elección del método se debe plantear en función de los tipos de imágenes a utilizar. Así, para imágenes homogéneas sin grandes cambios de iluminación es preferible la regresión lineal, mientras que para imágenes con mayor contraste, el adaptativo lineal. Siendo desaconsejable, en cualquier caso, el ajuste lineal ya sea de los valores máximos y mínimo de la imagen o

en función de la media y la desviación típica [23].

La elección del tipo de muestra es crucial, en cualquiera de los cuatro métodos expuestos anteriormente. El empleo de la imagen completa es inapropiado para el cálculo de la transformación. En cuanto al otro tipo de muestras, invariantes y pseudo-invariantes, los mejores resultados se obtienen con las muestras invariantes.

Como ya comentamos en la sección anterior, el objetivo de este proyecto es automatizar en la medida de lo posible el proceso de detección de cambios. El empleo de métodos sujetos a la elección de muestras específicas para la adecuada normalización de las imágenes, obliga a un alto grado de intervención humana, tanto para su selección como su posterior validación. Esto nos obliga a optar por otro tipo de técnicas que permitan automatizar la etapa de corrección radiométrica, minimizando la intervención humana.

Existe un quinto procedimiento, que a diferencia de los cuatro anteriores no realiza una corrección relativa entre imágenes a partir de muestras específicas, sino que calibra los valores digitales de las imágenes de modo absoluto, convirtiéndolos a medidas de *reflectividad*[6].

La conversión de los valores digitales almacenados en una imagen a variables físicas es un paso previo al establecimiento de modelos teóricos, así como a muchos estudios de tipo empírico. Ahora bien, además de servir para generar variables derivadas (temperaturas, índices de vegetación, etc.), el proceso de conversión tiene interés en sí mismo, pues facilita información física relevante, al proporcionar imágenes de reflectividad.

La reflectividad son variables físicas extrapolables a otras zonas y comparables entre sí, lo que hace más sólida la interpretación de los datos, garantiza la compatibilidad multi-temporal y el análisis integrado entre imágenes de distintos sensores [4].

Sin embargo, los procedimientos para abordar estas correcciones con precisión son muy laboriosos y requieren datos sobre las condiciones de la atmósfera en el momento de tomar la imagen, que no suelen estar disponibles. Además, el efecto de la dispersión atmosférica no es constante en la imagen, sino que determinadas zonas pueden haber sido más afectadas que otras, en

[6] **Reflectividad (ρ)**, relación entre el flujo incidente y el reflejado por una superficie.

función de la diversa presencia de aerosoles o vapor de agua.

6.3.2 Solución aportada

Según lo expuesto hasta el momento, la *automatización* del proceso de corrección radiométrica requiere consideraciones de diversa índole: la elección de un método de corrección independiente del contenido de las imágenes y que no requiera de la intervención humana para seleccionar muestras radiométricas representativas del desajuste entre imágenes.

Atendiendo a estas consideraciones, hemos optado por elegir una técnica que automatice la etapa de homogeneización radiométrica de las imágenes que intervienen en el proceso de detección de cambios y sea independiente del contenido de la imagen. La técnica empleada para tal fin es la *especificación del histograma*.

Especificación del histograma

La especificación del histograma permite transformar una imagen de tal forma que la imagen resultante posea un histograma determinado. Esta transformación se calcula a partir del histograma de la imagen original y del histograma especificado para la imagen resultante [10].

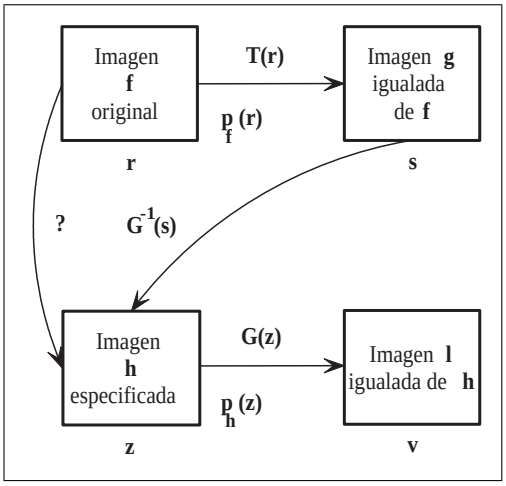


Figura 6.23: Esquema del proceso de especificación del histograma.

En nuestro caso, se toma una imagen como referencia, se calcula su his-

tograma y se especifica el histograma del resto con el histograma calculado[7].

Matemáticamente, la especificación del histograma vendría definida por las siguientes expresiones:

1. Igualar el histograma de la imagen a corregir:

$$s_k = T(r_k) = \sum_{j=0}^{k} p_f(r_j) \tag{6.15}$$

donde $p_f(r_j)$ es la frecuencia relativa de la intensidad r_j (histograma de la imagen a corregir).

> **NOTA:** Si suponemos inicialmente que r está normalizada y es continua en el intervalo $[0, 1]$. Para cualquier valor de intensidad r_k en dicho intervalo se cumple que:
>
> (a) $0 \leq T(r_k) \leq 1$ para $0 \leq r_k \leq 1$
>
> (b) $T(r_k)$ no está multivaluada
>
> (c) $T(r_k)$ es monótona creciente
>
> Así, la condición 1, obliga a que los nuevos valores de niveles de grises asignados estén en todo momento dentro del rango de intensidades permitidos. La condición 2, garantiza la asignación de un único valor de intensidad. Por último, la condición 3, impide que los niveles de gris de los píxeles de la imagen puedan invertirse entre ellos, con la consiguiente distorsión cromática.

2. Obtener la función de transformación $G(z_k)$ a partir del histograma de la imagen de referencia:

$$v_k = G(z_k) = \sum_{j=0}^{k} p_h(z_j) \tag{6.16}$$

donde $p_h(z_j)$ es la frecuencia relativa de la intensidad z_j (histograma de la imagen de referencia).

[7] El proceso de especificación se realiza dos a dos (imagen de referencia e imagen a corregir).

3. Aplicar la función de transformación inversa $G^{-1}(s_k)$ a las intensidades s_k de la imagen obtenida en el paso 1.

$$z_k = G^{-1}(s_k) = G^{-1}(T(r_k)) \tag{6.17}$$

Con esto finaliza el proceso de especificación del histograma, como resultado disponemos de una serie multi-temporal de imágenes convenientemente normalizadas radiométricamente, y sin haber sido necesaria ningún tipo de intervención humana.

6.3.3 Innovaciones

Básicamente, las innovaciones introducidas con esta técnica corrección radiométrica son las siguientes:

1. Empleamos una técnica normalización radiométrica que se caracteriza por lo siguiente:

   (a) Es independiente del contenido de la imagen.

   (b) No emplea ningún tipo de muestra representativa del desajuste radiométrico de las imágenes para realizar la corrección.

En resumen, normalizamos radiométricamente las imágenes que intervienen en el análisis multi-temporal, y lo hacemos sin ningún tipo de intervención humana. Con la ventaja de que esta técnica se puede acoplar perfectamente, a la etapa de corrección geométrica de tal forma que al finalizar ambas, disponemos de una serie multi-temporal de imágenes preparadas para realizar el proceso de transformación (diferencia de imágenes).

En las figuras 6.24, 6.25 y 6.26 podemos observar gráficamente la etapa de *corrección radiométrica*. También podemos observar como el histograma de la imagen corregida radiométricamente presenta un aspecto "quebrado", esto se debe al tamaño de las imágenes utilizadas (256×256 píxeles). Si el tamaño de las imágenes fuese mayor (por ejemplo, 1024×1024 píxeles) el histograma de la imagen corregida presentaría un aspecto mucho más parecido al de la imagen de referencia.

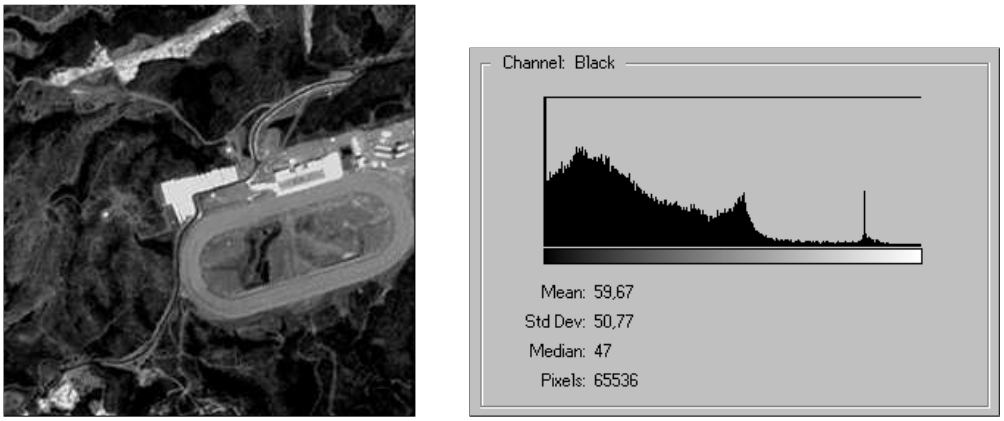


Figura 6.24: Histograma de la imagen de referencia.

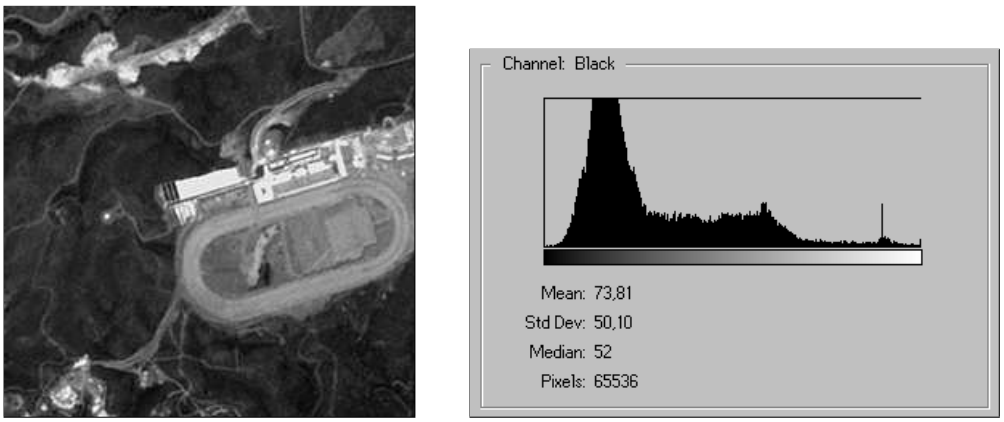


Figura 6.25: Histograma de la imagen a corregir.

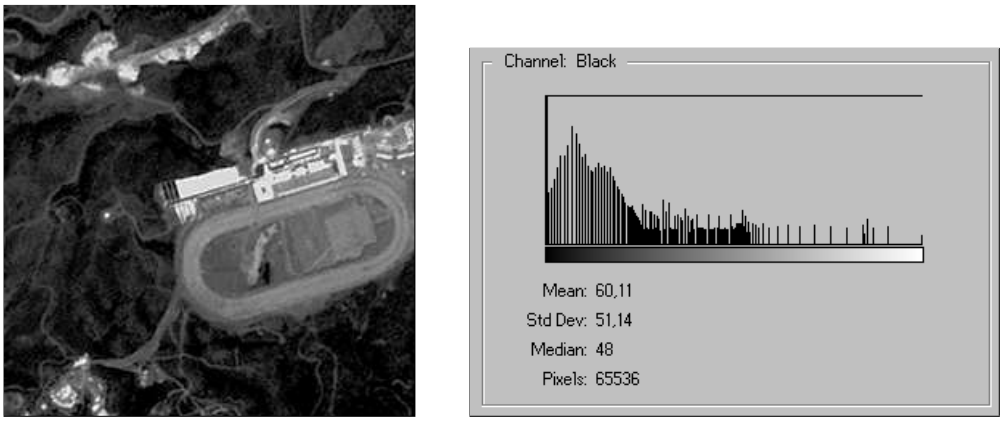


Figura 6.26: Histograma de la imagen corregida radiométricamente.

6.4 Detección de cambios

En las secciones 6.2 y 6.3 hemos tratado en profundidad las diferentes técnicas con las que se viene trabajando para realizar tanto la corrección geométrica como la radiométrica, y hemos optado por las técnicas que hemos creído más convenientes para afrontar con éxito ambas etapas. Por último, queda la etapa destinada a detectar los cambios que pudieran haberse producido durante el periodo de estudio.

6.4.1 Antecedentes

En los últimos años han proliferado notablemente los estudios de detección de cambios, aplicándose a una gran variedad de disciplinas [14]. Intentando resumir estos trabajos, podemos presentar las técnicas empleadas en dos grandes bloques, según utilicen imágenes continuas o categorizadas [5]. En el primer caso se emplean técnicas cuantitativas: diferencia, regresión, componentes principales, etc., mientras que en el segundo se comparan imágenes previamente clasificadas, mediante tablas de contingencia.

A continuación, vamos a repasar las distintas técnicas de detección, atendiendo únicamente a las cuantitativas. La detección de cambios puede abordarse de acuerdo a seis procedimientos [4]:

1. Composiciones multi-temporales

Esta técnica de detección se basa en comparar visualmente los tonos de gris o color que ofrecen dos o más imágenes de distinta fecha. Habitualmente, se aplican previamente algunas técnicas de realce del color o transformaciones, como índices de vegetación (NDVI) o componentes principales. Posteriormente se realizan composiciones multi-temporales en color con esas bandas [17]. Una muy común es utilizar el canal rojo de la primera fecha y el verde de la segunda, dejando el azul vacío. En amarillo aparecerán las zonas estables, en rojos las zonas que hayan reducido sus valores digitales entre fechas y en verde las que lo hayan ganado.

2. Diferencia entre imágenes

Una simple resta entre las imágenes de dos fechas, previamente homogeneiza-

das radiométrica y geométricamente, permite discriminar aquellas zonas que han experimentado cambios entre esas fechas. Las zonas estables presentarán un valor cercano a cero, mientras las que hayan experimentado cambios ofrecerán valores significativos distintos a cero (positivos o negativos):

$$ND_c = ND_{t2} - ND_{t1} + C \tag{6.18}$$

donde ND_c es el valor digital de la imagen de cambios, ND_{t1} y ND_{t2} los valores digitales de las imágenes correspondientes a la primera y segunda fecha y C una constante para evitar valores negativos (habitualmente 127).

3. Cocientes multi-temporales

La diferencia entre imágenes resulta una técnica sencilla para observar cambios entre fechas, aunque tiene el problema de reflejar únicamente las diferencias absolutas, no ofrece la importancia del cambio frente a los valores originales. Para subsanar este problema, puede ser más conveniente en ocasiones emplear cocientes multi-temporales, que ofrecen una valoración relativa del cambio:

$$ND_c = \frac{ND_{t2}}{ND_{t1}} * C \tag{6.19}$$

donde C es una constante para escalar el resultado (habitualmente 255).

4. Componentes principales

El análisis de componentes principales (ACP) permite sintetizar un conjunto de bandas en otro más reducido, sin perder gran parte de la información original. En el caso de aplicaciones multi-temporales se utiliza esta técnica de un modo particular. Se genera, en primer lugar, un archivo multi-temporal con las bandas correspondientes a las dos fechas, sobre el que se aplica el ACP. En este caso, los primeros componentes resultantes del análisis no son los más interesantes, ya que recogerán la información común a las dos fechas. Por el contrario, las componentes inferiores ofrecen la información no común: el cambio, que es lo más interesa en este contexto.

5. Regresión

Como es bien sabido, las técnicas de regresión se emplean para estimar valores de una variable de interés a partir de otra que está fuertemente asociada

con ella. Esa asociación se mide a partir de unas observaciones comunes a ambas variables, a partir de las cuales se ajusta una función que las relaciona numéricamente. La técnica de regresión más utilizada es la lineal simple, en la cual la variable a estimar (dependiente) se calcula a partir de otra auxiliar (independiente) con una ecuación del tipo:

$$N\hat{D}_{t2} = a + bND_{t1} \tag{6.20}$$

donde $N\hat{D}_{t2}$ es el valor digital estimado de la imagen correspondiente a la segunda fecha y, a y b los coeficientes de regresión.

Si se han producido cambios entre imágenes, los valores digitales reales de la segunda fecha presentarán valores alejados de los estimados por la regresión, o lo que es lo mismo contarán con residuos elevados:

$$ND_c = ND_{t2} - N\hat{D}_{t2} \tag{6.21}$$

6. Vectores multi-temporales

Es una técnica que intenta incorporar no sólo la importancia, sino también la dirección del cambio entre imágenes. Si representamos en un eje abscisas y ordenadas dos bandas originales, cada píxel viene definido por un punto[8]. Si ese píxel cambia su cobertura entre dos fechas, también modificará su emplazamiento espectral. La magnitud del cambio vendrá dado por la longitud del vector que separa ambos puntos.

$$d_{i,j,c} = \sqrt{(ND_{i,t1} - ND_{i,t2})^2 + (ND_{j,t1} - ND_{j,t2})^2} \tag{6.22}$$

donde i y j son las bandas originales y $d_{i,j,c}$ la intensidad del cambio espectral.

Por su parte, la dirección del cambio se define por el ángulo que forma con el eje de referencia.

$$\alpha = \arctan\left(\frac{ND_{j,t1} - ND_{j,t2}}{ND_{i,t1} - ND_{i,t2}}\right) \tag{6.23}$$

donde α es la dirección del cambio.

A continuación describimos con detalle el procedimiento seguido en este proyecto.

[8]Localización de su valor digital en las dos bandas.

Diferencia entre imágenes

La variedad de técnicas disponibles para detectar cambios entre imágenes es bastante importante, sin embargo, la complejidad de los procedimientos necesarios para llevar algunos de ellos, no compensan las resultados obtenidos. Puesto que básicamente, nuestro objetivo es localizar aquellas regiones de la imagen en la se han producidos cambios, hemos optado por la diferencia entre imágenes, que pese a su sencillez, nos permite obtener una imagen de cambios adecuada a nuestros propósitos.

Como comentamos anteriormente, la diferencia entre imágenes de dos fechas, previamente homogeneizadas radiométrica y geométricamente, permite discriminar aquellas zonas que han experimentado cambios entre esas fechas. Las zonas que no han sufrido cambios presentarán un valor cercano a cero, mientras las que hayan sufrido cambios ofrecerán valores distintos a cero (positivos o negativos).

La imagen de cambios presentará tonos oscuros para las zonas que hayan reducido su valor digital, los claros para aquéllas que hayan ganado y los intermedios para las zonas estables. El cálculo puede realizarse sobre los valores digitales de alguna de las bandas originales, aunque es frecuente aplicar esta operación sobre índices de vegetación (NDVI).

El histograma de esa imagen de cambios suele tener un perfil gaussiano (ver figura 6.27), con las máximas frecuencias para las zonas estables (centro del histograma) y las dinámicas ocupando las colas de la distribución (extremos del histograma).

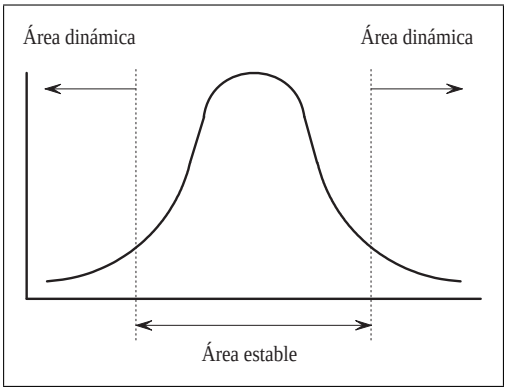


Figura 6.27: Histograma de la imagen de cambios, áreas estables y dinámicas.

6.4.2 Solución aportada

Según lo expuesto hasta el momento, la diferencia entre imágenes nos proporciona una imagen de cambios adecuada a nuestras necesidades. Si además de la imagen de cambios, fuese necesaria información relativa a la importancia del cambio detectado sería necesario emplear otra técnica de detección (por ejemplo, cocientes multi-temporales).

Uno de los aspectos importantes de esta técnica, y que cabría destacar, es el histograma tan característico que presenta la imagen de cambios. El estudio de este histograma nos permite determinar los umbrales que separan las áreas estables de las dinámicas (segmentación).

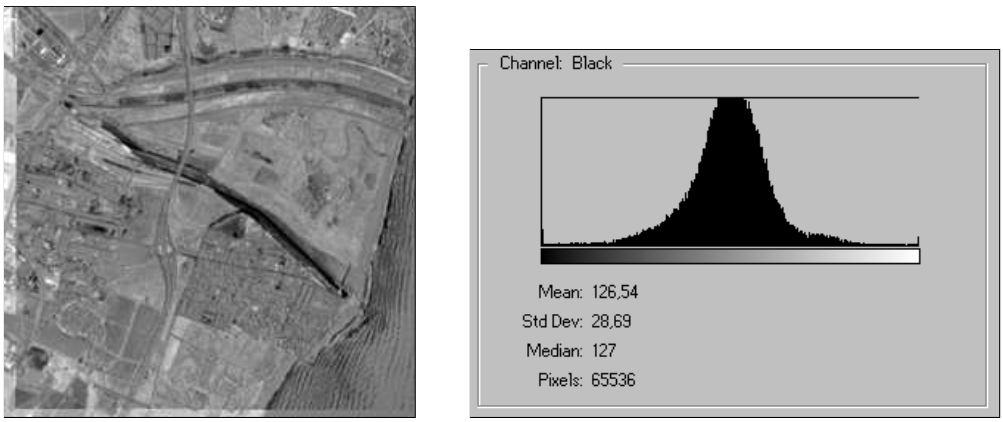


Figura 6.28: Histograma de la imagen de cambios $ND_c = ND_{t2} - ND_{t1} + C$.

Sin embargo, nos puede resultar útil, determinar la existencia de un cambio sin importar si es positivo o negativo. Es decir, nos puede bastar con detectar el cambio y no el sentido, por que tan ilegal es construir como derruir sin la correspondiente licencia municipal de obras. Para ello, modificaremos la expresión (6.18), quedando de la siguiente forma:

$$ND_c = |ND_{t2} - ND_{t1}| \tag{6.24}$$

El histograma de la imagen de cambios, presenta en este caso, un área dinámica y no dos, como ocurría en el caso anterior (ver figura 6.29).

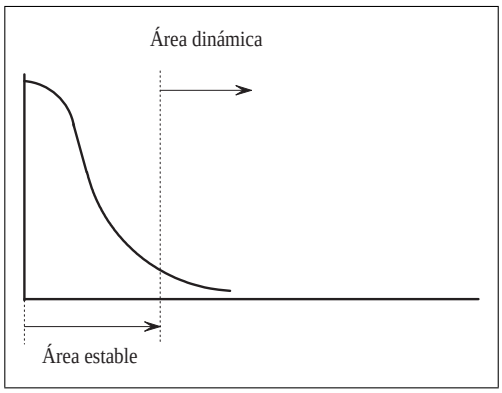


Figura 6.29: Histograma de la imagen de cambios, área estable y dinámica.

En este caso, bastaría con determinar el umbral adecuado para separar el área estable de la dinámica (segmentación).

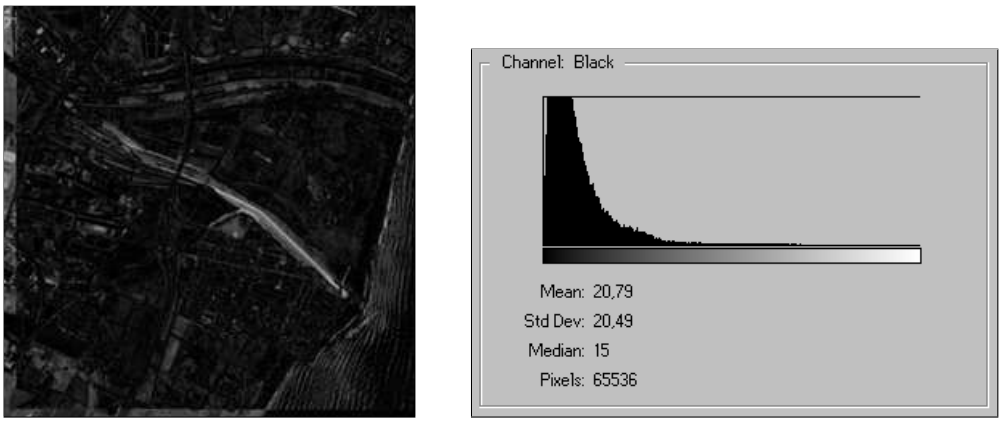


Figura 6.30: Histograma de la imagen de cambios $ND_c = |ND_{t2} - ND_{t1}|$.

6.4.3 Innovaciones

Las innovaciones introducidas con esta técnica de detección de cambios son las siguientes:

1. Empleamos una técnica sencilla, fácil de implementar y con unos resultados adecuados a nuestro propósito.

2. Los histogramas característicos que presentan las imágenes de cambios obtenidas facilitan la segmentación de la imagen.

3. Introducimos el concepto de sentido del cambio, que carece de interés para el proceso que pretendemos automatizar, pero que es de vital importancia, si se desea proporcionar información relativa a cuando se produjo el cambio.

Además de las ventajas que presenta la técnica descrita, también se caracteriza por que no requiere prácticamente ningún tipo de intervención humana. Lo que la hace especialmente útil para la tercera etapa de que consta el proceso de detección de cambios.

Capítulo 7

Análisis y diseño

7.1 Introducción

Una vez descritas las diferentes técnicas que serán utilizadas para realizar cada una de las etapas del proceso de detección de cambios, pasaremos a analizar y diseñar cada uno de los componentes software encargados de llevar a cabo dicho proceso.

Para realizar las tareas de análisis y diseño hemos empleado la técnica de modelado orientado a objetos UML (Unified Modeling Language) que consta básicamente de los siguientes modelos [11]:

1. *Conceptualización*[1]. Consiste en la primera aproximación al problema que se debe resolver: Casos de uso.

2. *Modelo de objetos*. Describe la estructura estática de los objetos de un sistema y sus relaciones: Diagramas de clases.

3. *Modelo dinámico*. Describe las características de un sistema que cambia a lo largo del tiempo: Diagramas de estados.

4. *Modelo funcional*: Describe las transformaciones de datos del sistema. Diagramas de interacción y especificación de operaciones.

UML reúne las mejores características de los tres métodos más utilizados de modelado orientado a objetos (Booch, OMT, and OOSE) e incluye expresiones adicionales para modelar problemas que estos no contemplaban.

[1] No es un modelo, pero facilita las tareas de modelado.

7.2 Conceptualización

Como comentamos anteriormente, la etapa de conceptualización no constituye en sí misma un modelo del sistema, pero proporciona una primera aproximación al problema que pretendemos resolver. Para realizar las tareas de conceptualización emplearemos *casos de uso*.

7.2.1 Casos de uso (CU)

Como podemos observar en la figura 7.1, el proceso de detección de cambios requiere la ejecución secuencial de tres etapas claramente diferenciadas:

1. Corrección geométrica
2. Corrección radiométrica
3. Diferencia de imágenes

El programa (o programas) debe aceptar datos del usuario (imágenes y datos colaterales) y proporcionar los datos procesados en un formato entendible por el usuario.

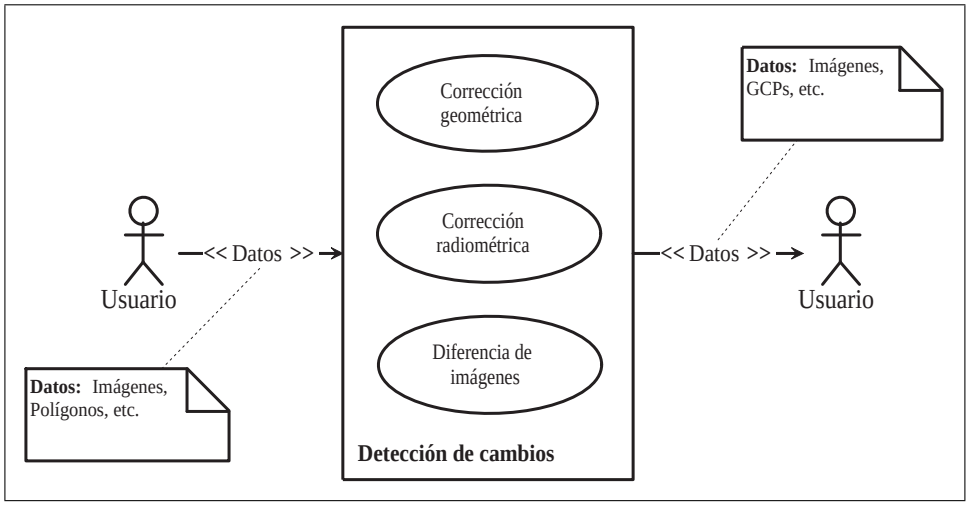


Figura 7.1: CU del sistema.

7.3 Modelo de objetos

El modelo de objetos describe la estructura de los objetos de un sistema: su identidad, sus relaciones, sus atributos y sus operaciones.

El modelo de objetos se representa gráficamente con diagramas de objetos e instancias, que contienen clases de objetos e instancias, respectivamente. Las clases se disponen en jerarquías que comparten una estructura y comportamiento comunes, y se relacionan con otras clases. Cada clase define los atributos que contiene cada uno de los objetos o instancias y las operaciones que realizan.

7.3.1 Diagrama de clases (DC)

El diagrama consta de ocho clases:

1. `Kernel`
2. `Image`
3. `MImage`
4. `DImage`
5. `ImageInfo`
6. `Matrix<double>`. Instanciación de `Matrix<Tipo numérico>`
7. `Polygon`
8. `FastFormat`

La clase `Kernel` centraliza la mayor parte del procesamiento. Proporciona las operaciones asociadas a las etapas del proceso de detección, gestiona el fichero de registro, etc. Las clases `Image` (y sus hijas `MImage` y `DImage`) y `ImageInfo`, encapsulan los atributos y operaciones propias de una imagen. El resto son clases de apoyo: `Polygon` y `Matrix<Tipo numérico>` para la corrección geométrica y `FastFormat` para la lectura del fichero de cabecera *Fast Format Rev. C* de las imágenes de satélite.

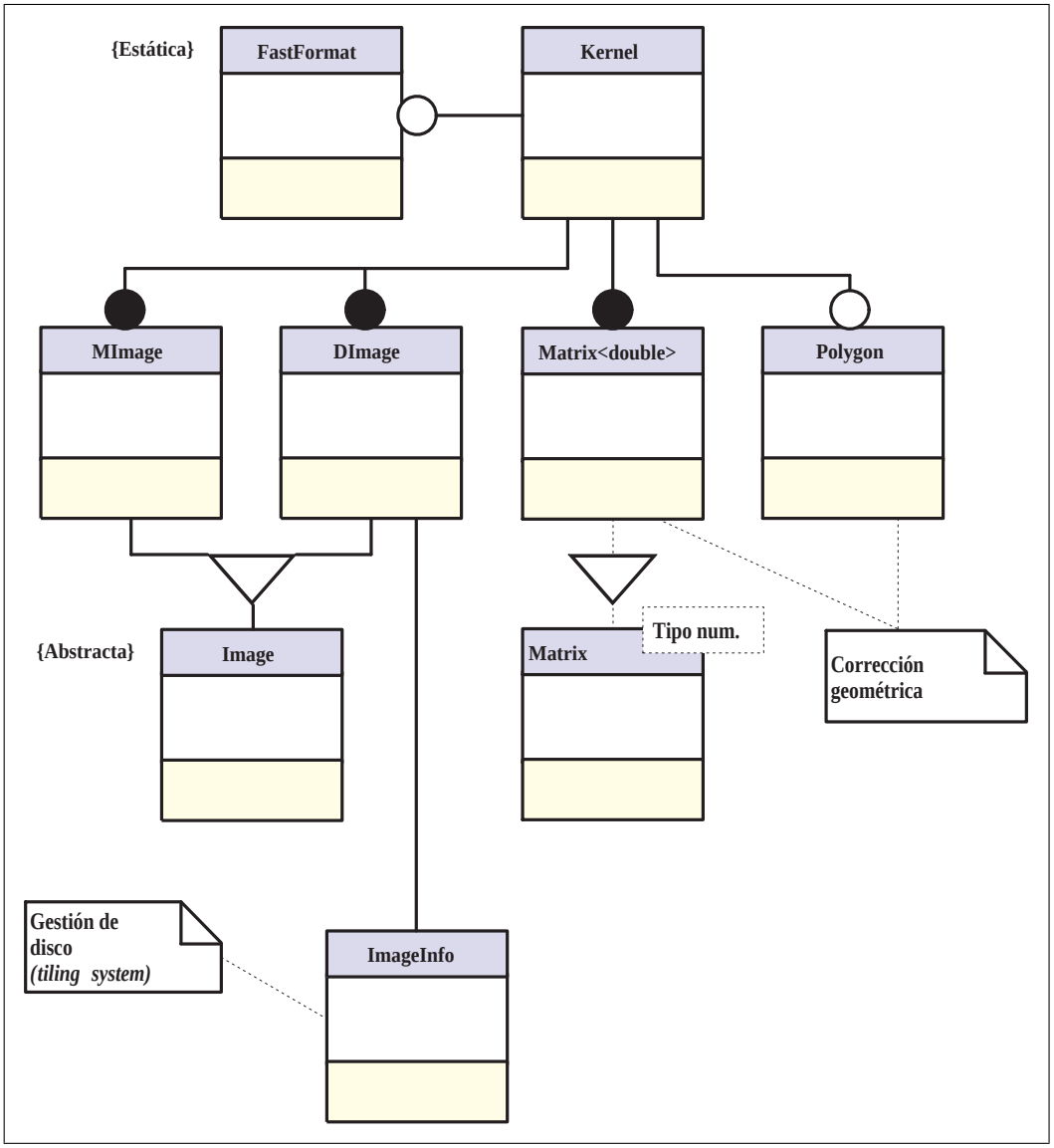


Figura 7.2: DC del sistema.

NOTA: La descripción de las clases, se centrará en los atributos y operaciones relevantes. Para un listado detallado de las clases, atributos y operaciones, ver el apéndice A.

Kernel

La clase `Kernel` (ver figura 7.3) encapsula las operaciones asociadas a las etapas del proceso de detección de cambios: corrección geométrica, corrección radiométrica y diferencia de imágenes. Gestiona el fichero de registro, en el que se almacena información de interés y el resultado de las diferentes etapas, etc.

Para implementar algunas de las operaciones de esta clase hemos utilizado las funciones y estructuras proporcionadas por el *KLT Feature Tracker 1.1.5* y el *USGS General Cartographic Transformation Package 2.0*.

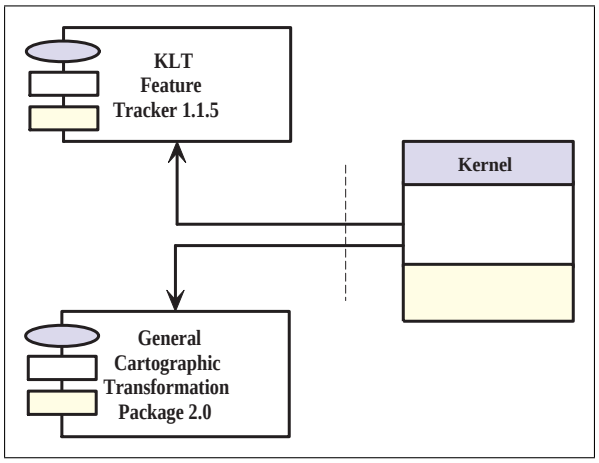


Figura 7.3: Clase `Kernel`.

Atributos

– `Mode:int` : Tipo de procesamiento de la imagen de cambios:

1. Valor absoluto (*Absolute*)
2. Offset (*Offset*)

– `TileWidth:int` y `TileHeight:int` : Dimensiones del tile utilizado en el proceso de detección y seguimiento de GCPs

– `LogFile:file` : Fichero de registro

Operaciones

– `GeometricCorrection(Image,Image):Image`

- *Argumentos:*
  1. `Image` : Imagen de referencia
  2. `Image` : Imagen a corregir
- *Función:* Corrige geométricamente las imágenes especificadas
- *Resultado:*
  1. `Image` : Imagen corregida

– `RadiometricCorrection(Image,Image):Image`

- *Argumentos:*
  1. `Image` : Imagen de referencia
  2. `Image` : Imagen a corregir
- *Función:* Corrige radiométricamente las imágenes especificadas
- *Resultado:*
  1. `Image` : Imagen corregida

– `ChangesDetection(Image,Image):Image`

- *Argumentos:*
  1. `Image` : Imagen de referencia
  2. `Image` : Imagen a corregir
- *Función:* Resta las imágenes especificadas y procesa la imagen diferencia (valor absoluto u offset)
- *Resultado:*
  1. `Image` : Imagen diferencia procesada

– `ComputeCoefficients(Matrix<double>,Matrix<double>):Matrix<double>`

- *Argumentos:*
  1. `Matrix<double>` : Coordenadas de los GCPs de la imagen de referencia
  2. `Matrix<double>` : Coordenadas de los GCPs de la imagen a corregir
- *Función:* Calcula los coeficientes del polinomio de ajuste a partir de las coordenadas especificadas
- *Resultado:*
  1. `Matrix<double>` : Coeficientes calculados

– `ComputeResiduals(Matrix<double>,Matrix<double>,Matrix<double>):int`

- *Argumentos:*
  1. `Matrix<double>` : Coordenadas de los GCPs de la imagen de referencia
  2. `Matrix<double>` : Coordenadas de los GCPs de la imagen a corregir
  3. `Matrix<double>` : Coeficientes del polinomio de ajuste
- *Función:* Selecciona, en función de error cuadrático medio (RMS), el GCP candidato para su eliminación (política de selección)
- *Resultado:*
  1. `int` : Punto seleccionado

Image

La clase abstracta `Image` (ver figura 7.4) encapsula los atributos y operaciones propios de una imagen.

Para implementar algunas de las operaciones de esta clase hemos utilizado las funciones y estructuras proporcionadas por la *Intel Image Processing Library 2.5*.

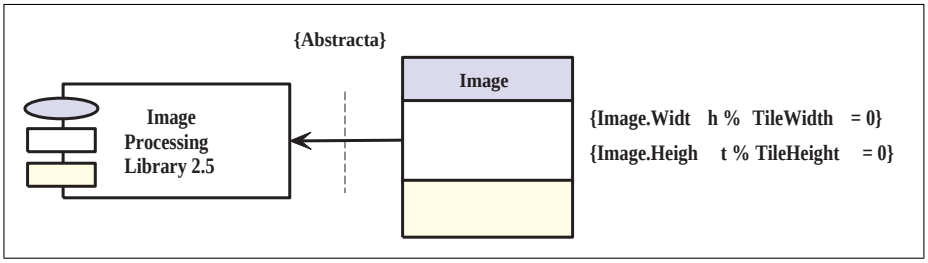


Figura 7.4: Clase `Image`.

Atributos

– `Levels:int` : Niveles de gris de las imágenes utilizadas

– `TileHeight:int` y `TileWidth:int` : Dimensiones del tile utilizado en el proceso de detección y seguimiento de GCPs

– `Image:IplImage` : Estructura de la *Intel Image Processing Library 2.5* que encapsula los atributos propios de una imagen:

- Alto y ancho
- Canales (GRAY, RGB, etc.)
- Formato (unsigned char, float, etc.)

Operaciones

– `WarpBilinear(double[][]):Image`

- *Argumentos:*
  1. `double[][]` : Coeficientes del polinomio de ajuste
- *Función:* Realiza la convolución cúbica de la imagen receptora a partir de los coeficientes especificados
- *Resultado:*
  1. `Image` : Imagen procesada

– `ComputeHistogram():int[]`

- *Argumentos:* Ninguno
- *Función:* Calcula el histograma de la imagen receptora
- *Resultado:*
  1. `int[]` : Histograma calculado

– `ContrastStretch(int[])`

- *Argumentos:*
  1. `int[]` : Histograma especificado
- *Función:* Aplica el histograma especificado a la imagen receptora
- *Resultado:* Ninguno

– `EqualizeHistogram():int[]`

- *Argumentos:* Ninguno
- *Función:* Ecualiza el histograma de la imagen receptora
- *Resultado:*
  1. `int[]` : Histograma de la imagen ecualizada

– `Subtract_Abs(Image):Image`

- *Argumentos:*
  1. `Image` : Imagen a corregir
- *Función:* Resta la imagen especificada a la imagen receptora y calcula el valor absoluto de la imagen diferencia
- *Resultado:*
  1. `Image` : Imagen diferencia procesada

– `Subtract_Offset(Image,int):Image`

- *Argumentos:*
  1. `Image` : Imagen a corregir
  2. `int` : Desplazamiento
- *Función:* Resta la imagen especificada a la imagen receptora y desplaza la imagen diferencia un determinado nivel de gris
- *Resultado:*
  1. `Image` : Imagen diferencia procesada

– `LoadRAWFile(char[]) = 0` (Abstracta)

- *Argumentos:*
  1. `char[]` : Nombre del fichero (formato RAW)
- *Función:* Carga la imagen almacenada en el fichero especificado
- *Resultado:* Ninguno

– `SaveRAWFile(char[]) = 0` (Abstracta)

- *Argumentos:*
  1. `char[]` : Nombre del fichero
- *Función:* Salva la imagen en el fichero especificado (formato RAW)
- *Resultado:* Ninguno

MImage

La clase `MImage` (ver figura 7.5) hereda de `Image`.

La clase `MImage` ha sido desarrollada para trabajar con imágenes almacenadas en memoria. Este hecho nos permite trabajar eficientemente en sistemas que disponen de gran cantidad de memoria[2].

[2]Recordar que el tamaño habitual de las imágenes de trabajo oscila entre 1 y 25 Mb.

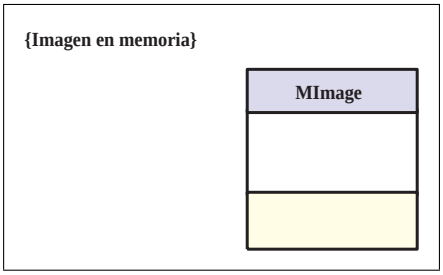


Figura 7.5: Clase `MImage`.

NOTA: `MImage` define las operaciones `LoadRAWFile` y `SaveRAWFile` de `Image`.

Operaciones

– `LoadRAWFile(char[])`

- *Argumentos:*
  1. `char[]` : Nombre del fichero (formato RAW)
- *Función:* Carga la imagen almacenada en el fichero especificado
- *Resultado:* Ninguno

– `SaveRAWFile(char[])`

- *Argumentos:*
  1. `char[]` : Nombre del fichero
- *Función:* Salva la imagen en el fichero especificado (formato RAW)
- *Resultado:* Ninguno

– `PutPixel(int,int,byte)`

- *Argumentos:*
  1. `int` : Fila

2. `int` : Columna
3. `byte` : Nivel de gris

- *Función:* Escribe el nivel de gris especificado en el pixel especificado
- *Resultado:* Ninguno

– `GetPixel(int,int):byte)`

- *Argumentos:*
  1. `int` : Fila
  2. `int` : Columna
- *Función:* Lee el nivel de gris del pixel especificado
- *Resultado:*
  1. `byte` : Nivel de gris

DImage

La clase `DImage` (ver figura 7.6) hereda de `Image`.

La clase `DImage` ha sido desarrollada para trabajar con imágenes almacenadas en disco. Este hecho nos permite trabajar en sistemas que disponen de una cantidad limitada de memoria.

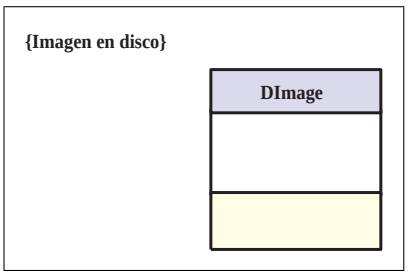


Figura 7.6: Clase `DImage`.

NOTA: `DImage` define las operaciones `LoadRAWFile` y `SaveRAWFile` de `Image`.

Atributos

– `Files:static int` : Número de ficheros temporales en disco

Operaciones

– `LoadRAWFile(char[])`

- *Argumentos:*
  1. `char[]` : Nombre del fichero (formato RAW)
- *Función:* Carga la imagen almacenada en el fichero especificado
- *Resultado:* Ninguno

– `SaveRAWFile(char[])`

- *Argumentos:*
  1. `char[]` : Nombre del fichero
- *Función:* Salva la imagen en el fichero especificado (formato RAW)
- *Resultado:* Ninguno

ImageInfo

La clase `ImageInfo` (ver figura 7.7) encapsula los atributos y operaciones necesarios para la gestión de disco (*tiling system*).

Para implementar algunas de las operaciones de esta clase hemos utilizado las funciones y estructuras proporcionadas por la *Intel Image Processing Library 2.5*.

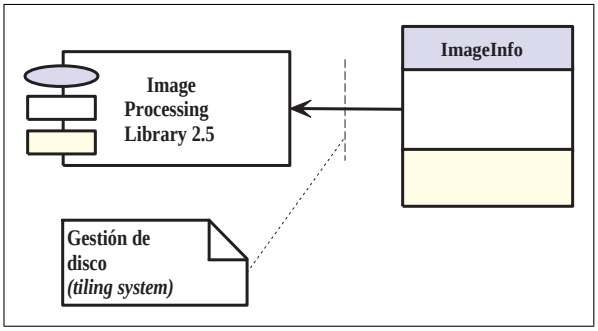


Figura 7.7: Clase `ImageInfo`.

Atributos

– `Width:int` y `Height:int` : Dimensiones de la imagen almacenada en disco

– `File:file` : Fichero temporal en disco

Operaciones

– `GetTileToRead(int,int):IplImage`

- *Argumentos:*
  1. `int` : Fila
  2. `int` : Columna
- *Función:* Lee un tile de la posición especificada para lectura
- *Resultado:*
  1. `IplImage` : Tile (contiene información)

– `GetTileToWrite():IplImage`

- *Argumentos:* Ninguno
- *Función:* Lee un tile de la posición especificada para escritura
- *Resultado:*
  1. `IplImage` : Tile (vacío)

– `ReleaseTile(int,int,IplImage)`

- *Argumentos:*
  1. `int` : Fila
  2. `int` : Columna
  3. `IplImage` : Tile
- *Función:* Escribe el tile especificado en la posición especificada
- *Resultado:* Ninguno

– `CallBack(int,int,int):IplImage`

- *Argumentos:*
  1. `int` : Fila
  2. `int` : Columna
  3. `int` : Modo (GetTileToRead, GetTileToWrite y ReleaseTile)
- *Función:* Función *callback* del sistema de *tiling* de la *Intel Image Processing Library 2.5*
- *Resultado:*
  1. `IplImage` : Vacío o el tile leído

Matrix<Tipo numérico>

El template `Matrix<Tipo numérico>` (ver figura 7.8) encapsula los atributos y operaciones propios de una matriz numérica, atendiendo fundamentalmente a las operaciones útiles para el cálculo determinantes, submatrices, etc.

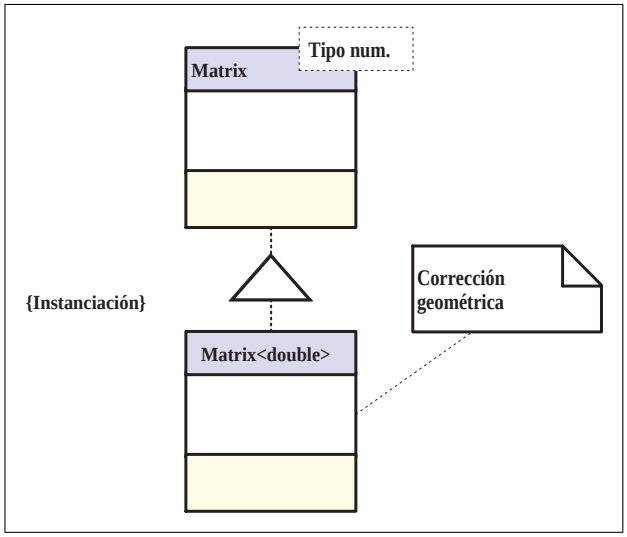


Figura 7.8: Template `Matrix<Tipo numérico>`.

Las matrices utilizadas en la clase `Kernel`, concretamente en la operación `ComputeCoefficients(Matrix,Matrix):Matrix`, son de tipo `double` (precisión subpíxel).

Atributos

– `Rows:int` y `Cols:int` : Dimensiones de la matriz numérica

– `Data:Tipo[]` : Array de datos numéricos

Operaciones

– `Determinant():Tipo`

- *Argumentos:* Ninguno
- *Función:* Calcula, si es posible, el determinante de la matriz
- *Resultado:*
    1. `Tipo` : Determinante

– `SubMatrix(int,int):Matrix`

- *Argumentos:*

1. `int` : Fila
2. `int` : Columna

- *Función:* Matrix construida eliminando la fila y columna especificadas de la matriz receptora
- *Resultado:*
  1. `Matrix` : Submatriz

– `RemoveCol(int)`

- *Argumentos:*
  1. `int` : Columna
- *Función:* Elimina la columna especificada de la matriz receptora
- *Resultado:* Ninguno

– `RemoveRow(int)`

- *Argumentos:*
  1. `int` : Fila
- *Función:* Elimina la fila especificada de la matriz receptora
- *Resultado:* Ninguno

Polygon

La clase `Polygon` (ver figura 7.9) encapsula los atributos y operaciones propios de un polígono cerrado, atendiendo fundamentalmente a las operaciones útiles para determinar si un punto (o conjunto de ellos) está en su interior (incluido su borde).

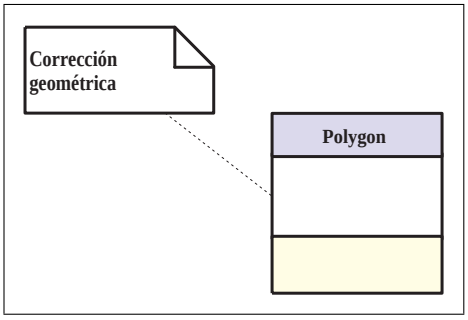


Figura 7.9: Clase `Polygon`.

Atributos

– `Size:int` : Número de vértices del polígono

– `Polygon:point[]` : Coordenadas de los vértices del polígono

Operaciones

– `PointInPolygon(int,int):bool`

- *Argumentos:*
  1. `int` : Fila
  2. `int` : Columna
- *Función:* Comprueba si el pixel situado en la posición especificada está dentro del polígono
- *Resultado:*
  1. `bool` : Verdadero si está dentro y falso en caso contrario

– `RectInPolygon(int,int,int,int):bool`

- *Argumentos:*
  1. `int` : Fila
  2. `int` : Columna
  3. `int` : Ancho

4. `int` : Alto

- *Función:* Comprueba si alguno de los vértices (píxeles) del rectángulo especificado está dentro del polígono
- *Resultado:*

1. `bool` : Verdadero si alguno está dentro y falso en caso contrario

FastFormat

La clase estática `FastFormat` (ver figura 7.10) encapsula los atributos y operaciones necesarios para recuperar fácilmente los distintos datos almacenados en el fichero de cabecera *Fast Format Rev. C* que acompaña a las imágenes de satélite utilizadas.

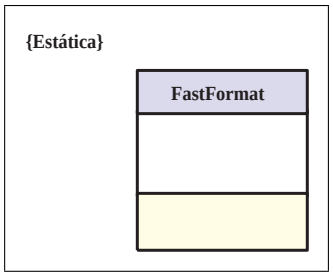


Figura 7.10: Clase `FastFormat`.

Atributos

– `ARField:static int[][]` : Matriz de campos (byte inicial y final) del registro administrativo

– `RRField:static int[][]` : Matriz de campos (byte inicial y final) del registro radiométrico

– `GRField:static int[][]` : Matriz de campos (byte inicial y final) del registro geométrico

– `File:file` : Fichero de cabecera

Operaciones

– `static GetARField(file,int):char[]`

- *Argumentos:*
  1. `file` : Fichero de cabecera
  2. `int` : Campo del fichero de cabecera
- *Función:* Devuelve el contenido del campo especificado del registro administrativo
- *Resultado:*
  1. `char[]` : Contenido del campo contrario

– `static GetRRField(file,int):char[]`

- *Argumentos:*
  1. `file` : Fichero de cabecera
  2. `int` : Campo del fichero de cabecera
- *Función:* Devuelve el contenido del campo especificado del registro radiométrico
- *Resultado:*
  1. `char[]` : Contenido del campo contrario

– `static GetGRField(file,int):char[]`

- *Argumentos:*
  1. `file` : Fichero de cabecera
  2. `int` : Campo del fichero de cabecera
- *Función:* Devuelve el contenido del campo especificado del registro geométrico
- *Resultado:*
  1. `char[]` : Contenido del campo contrario

7.3.2 Diagrama de componentes software (DCS)

Muchos de las técnicas a las que hacíamos referencia en el capítulo 6 se encuentran ya implementadas, y optimizadas para los equipos informáticos[3] disponibles actualmente. El objeto de este diagrama es mostrar las dependencias que se dan entre componentes software.

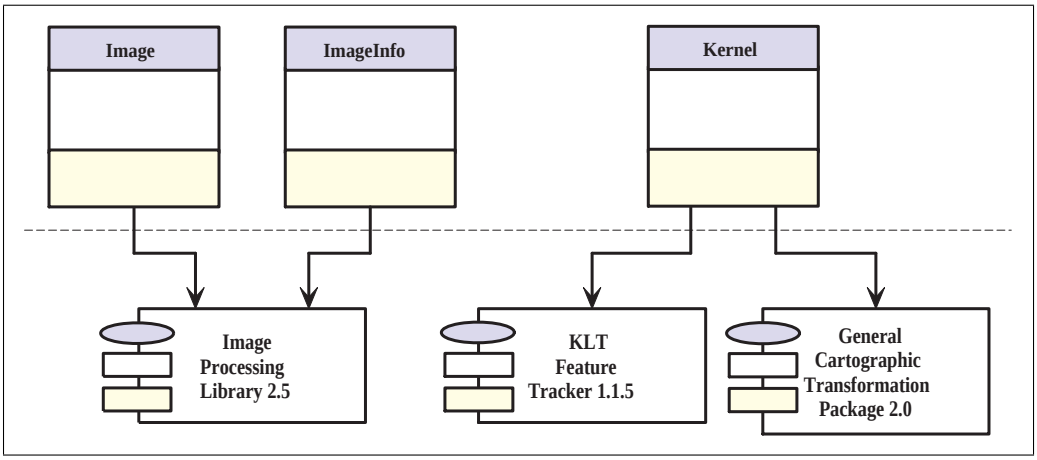


Figura 7.11: DCS del sistema.

Como podemos observar en la figura 7.11, nuestro diagrama de clases depende de tres paquetes software:

1. Intel Image Processing Library 2.5
   (`ftp://download.intel.com/software/`)

2. KLT Feature Tracker 1.1.5
   (`http://vision.stanford.edu/~birch/klt/`)

3. USGS General Cartographic Transformation Package 2.0
   (`ftp://edcftp.cr.usgs.gov/pub/software/gctpc/`)

El primero de ellos proporciona funciones para el procesamiento de imágenes, el segundo implementa el detector y seguidor de características, y el tercero funciones para la transformación de coordenadas entre diferentes tipos de proyecciones cartográficas.

[3]Por ejemplo, la gama de microprocesadores Intel.

7.4 Modelo dinámico

El modelo dinámico describe las características de un sistema que cambia a lo largo del tiempo. Se utiliza para especificar los aspectos de control de un sistema. Para representarlo utilizaremos diagramas de estados.

Al igual que todas las instancias de una clase comparten atributos, pero tienen valores particulares para esos atributos, todas las instancias de una clases se comportan de igual forma, pero pueden encontrarse en estados distintos.

7.4.1 Diagramas de estado (DE)

Debido a la notable complejidad de la clase `Kernel` centraremos nuestro estudio en esta clase, el comportamiento del resto de las clases es trivial y no precisa de diagramas de estados para representarlo. Además, para facilitar la comprensión, desglosaremos el diagrama de estados inicial en tres, asociados cada uno de ellos a las tareas fundamentales de esta clase: corrección geométrica, corrección radiométrica y diferencia de imágenes.

Clase Kernel

Como hemos comentado anteriormente, los diagramas que mostraremos a continuación corresponden a un mismo diagrama que ha sido desglosado en tres, para facilitar la compresión.

Corrección geométrica

Número de estados: 3

1. *Inactivo*
2. *Compute coefficients:* Calcula los coeficientes del polinomio de ajuste
3. *Warp bilinear:* Realiza la convolución cúbica de la imagen a partir de los coeficientes calculados

Número de transiciones: 5

1. Inactivo → Compute coefficients (Acción implícita: Compute GCPs)
2. Compute coefficients → Compute coefficients[4]
3. Compute coefficients → Warp bilinear
4. Warp bilinear → Inactivo

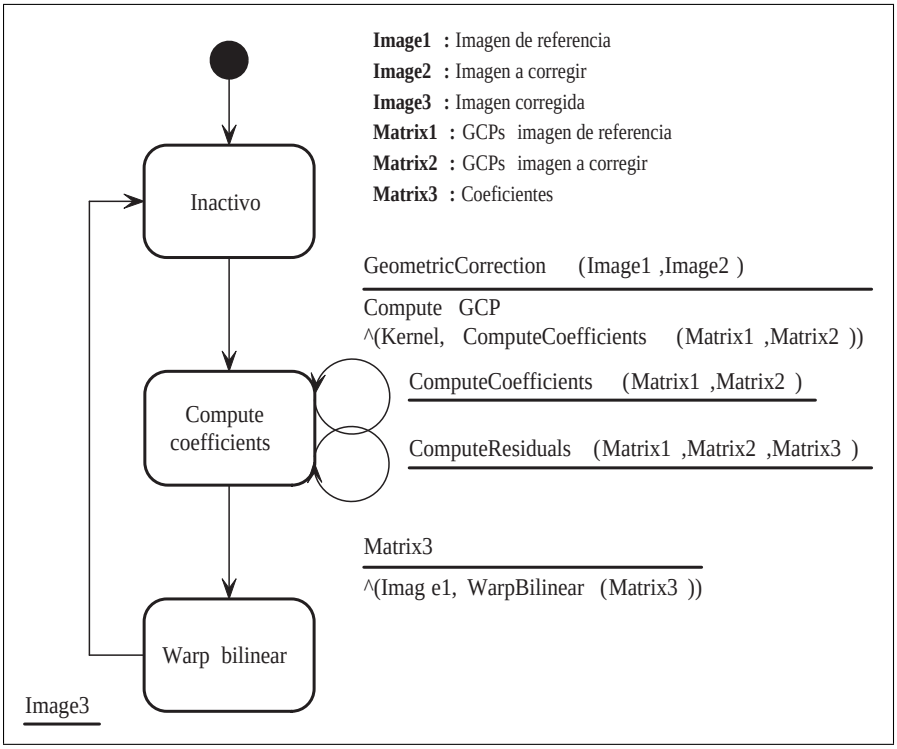


Figura 7.12: DE de la clase Kernel - Corrección geométrica.

Corrección radiométrica

Número de estados: 4

1. *Inactivo*
2. *Compute histogram:* Calcula el histograma de la imagen de referencia

[4]Transiciones cíclicas: Eliminación de GCPs incorrectos y cálculo de los nuevos coeficientes.

3. *Equalize histogram:* Ecualiza el histograma de la imagen a corregir
4. *Contrast stretch:* Especifica el histograma de la imagen ecualizada

Número de transiciones: 4

1. Inactivo → Compute histogram
2. Compute histogram → Equalize histogram
3. Equalize histogram → Contrast stretch (Acción implícita: Compute G^{-1})
4. Contrast stretch → Inactivo

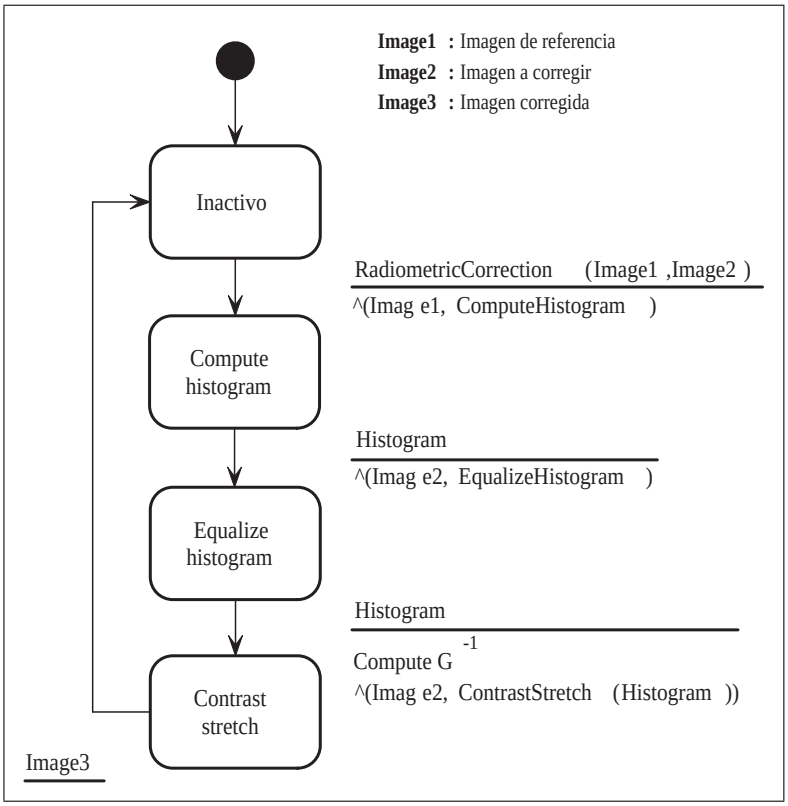


Figura 7.13: DE de la clase Kernel - Corrección radiométrica.

Diferencia de imágenes

Número de estados: 3

1. *Inactivo*
2. *Subtract and Abs:* Resta la imagen a corregir a la imagen de diferencia y calcula el valor absoluto de la imagen diferencia
3. *Subtract and Offset:* Resta la imagen a corregir a la imagen de diferencia y desplaza la imagen diferencia un determinado nivel de gris (offset)

Número de transiciones: 4

1. Inactivo → Subtract and Abs
2. Subtract and Abs → Inactivo
3. Inactivo → Subtract and Offset
4. Subtract and Offset → Inactivo

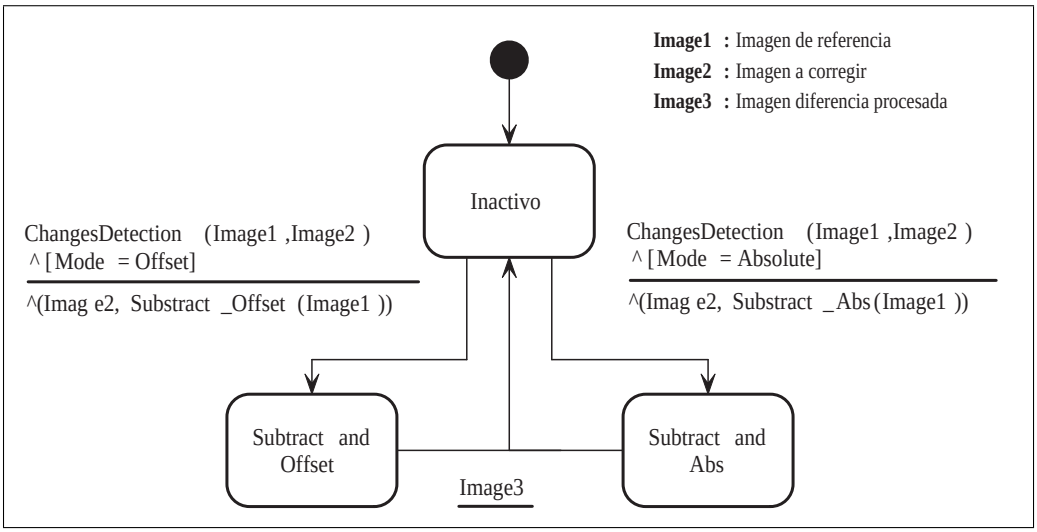


Figura 7.14: DE de la clase Kernel - Diferencia de imágenes.

7.5 Modelo funcional

El modelo funcional describe las computaciones que se realizan en un sistema, mostrando cómo se derivan los valores de salida a partir de los de entrada. Para realizar el modelo funcional del sistema se pueden utilizar casos de uso, especificación de operaciones y diagramas de interacción entre objetos.

7.5.1 Diagramas de interacción (DI)

Como ocurría en el modelo dinámico, debido a la notable complejidad de la clase `Kernel` centraremos nuestro estudio en esta clase, atendiendo fundamentalmente a las operaciones asociadas a las tareas del proceso de detección cambios.

Corrección geométrica

En el DI de la figura 7.15 podemos observar como interactúan dos objetos de las clases `Kernel` y `Image` respectivamente, para realizar la etapa de corrección geométrica del proceso de detección. De igual forma, podemos observar el flujo de datos entre objetos: GCPs, coeficientes de ajuste, etc.

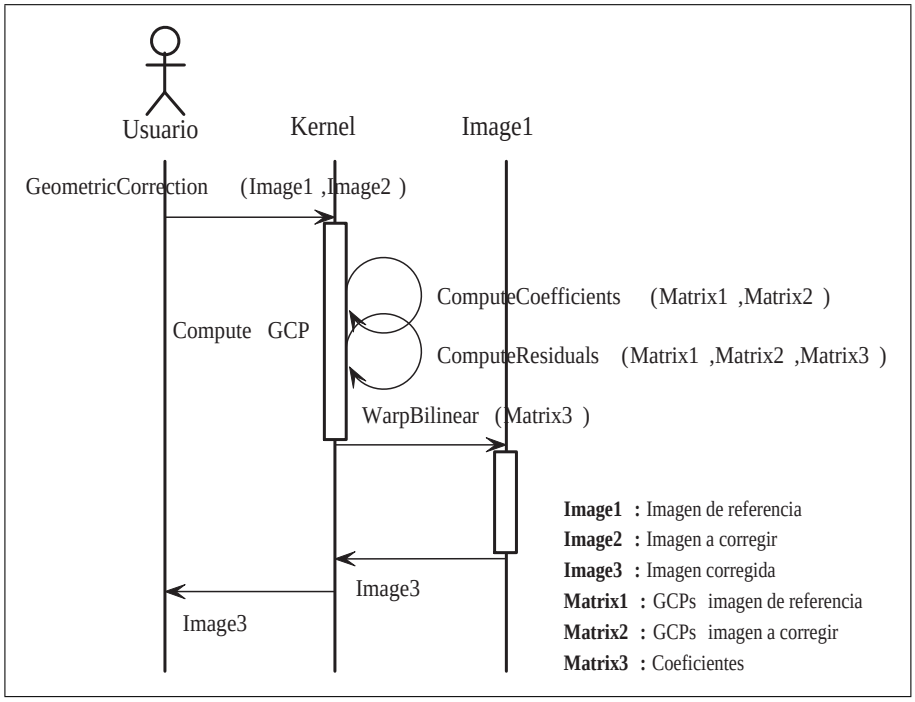


Figura 7.15: DI corrección geométrica.

Corrección radiométrica

En el DI de la figura 7.16 podemos observar como interactúan tres objetos de las clases `Kernel` y `Image` respectivamente, para realizar la etapa de co-

rrección radiométrica del proceso de detección. De igual forma, podemos observar el flujo de datos entre objetos: histogramas, funciones, etc.

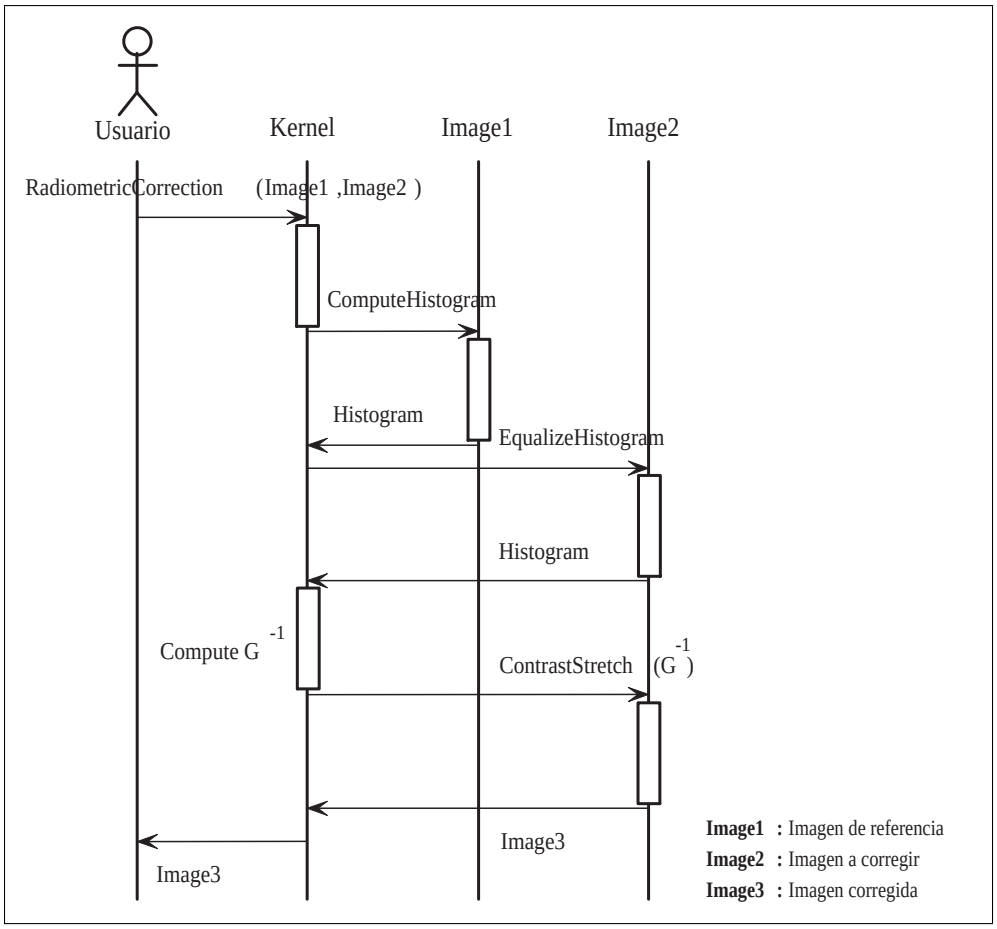


Figura 7.16: DI corrección radiométrica.

Diferencia de imágenes

En el DI de la figura 7.17 podemos observar como interactúan dos objetos de las clases `Kernel` y `Image` respectivamente, para realizar la etapa de diferencia de imágenes del proceso de detección. De igual forma, podemos observar el flujo de datos entre objetos: imagen diferencia.

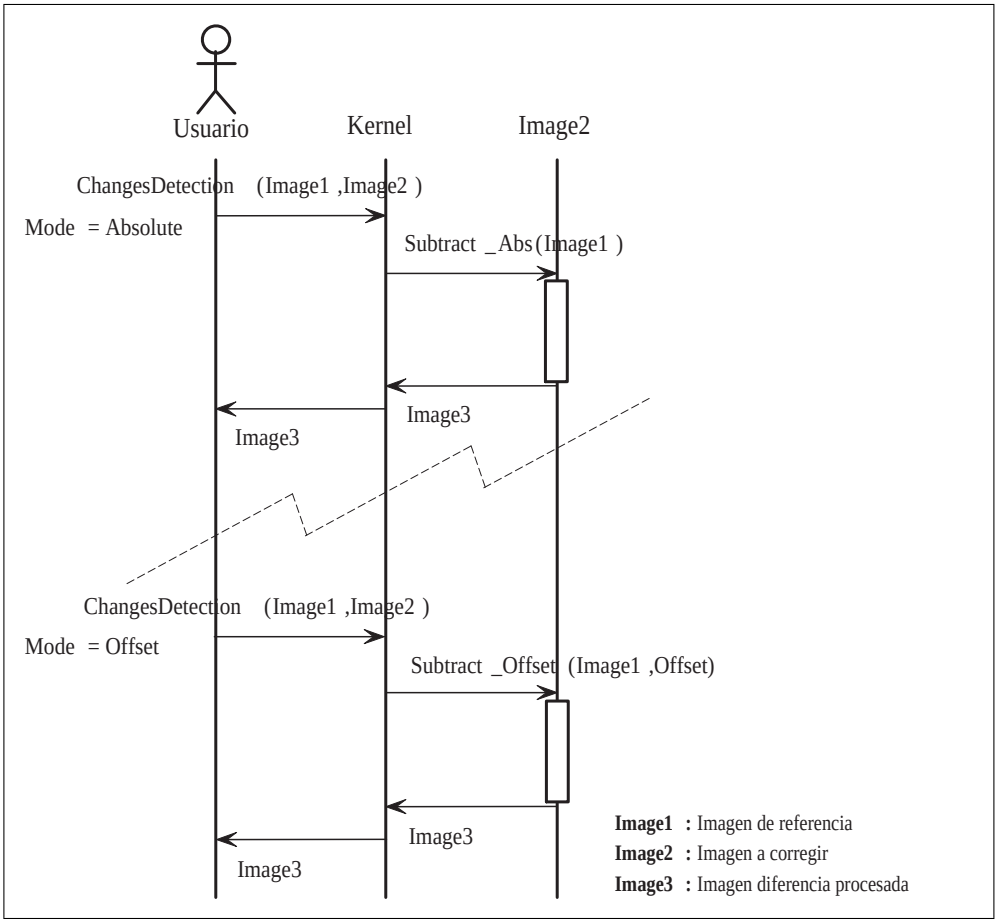


Figura 7.17: DI diferencia de imágenes.

7.5.2 Especificación de operaciones

Especificaremos las operaciones relevantes, tanto de la clase `Kernel` como de las clases relacionadas con ésta, es decir, las clases `Image`, Matrix<double> y `Polygon`, pero siempre desde el punto de vista de las tareas del proceso de detección.

> **NOTA:** Para realizar la especificación de las operaciones hemos utilizado pseudocódigo.

Clase Kernel

– `GeometricCorrection(Image,Image):Image`

```
/*
////////////////////////////////////////////////////////////////////
// Name:        GeometricCorrection
// Purpose:     Performs the geometric correction
// Context:     i42aresv Soft
// Returns:     Image
// Parameters:
        ImageR:Image - Reference image
        ImageC:Image - Image to correct
// Notes:
*/
Kernel::GeometricCorrection(ImageR:Image, ImageC:Image):Image
{
//  Number of tiles
    XTiles:int = (ImageR.GetWidth() / TileWidth);     //  Horizontal
    YTiles:int = (ImageR.GetHeight() / TileHeight);   //  Vertical

    Features: int = 25;

//  KLT environment
    tc:KLT_TrackingContext;

    flR:KLT_FeatureList;
    flC:KLT_FeatureList;

    KLTSetVerbosity(0);

    tc = KLTCreateTrackingContext();
    tc->window_width  = 15;
    tc->window_height = 15;
    KLTUpdateTCBorder(tc);

    flR = KLTCreateFeatureList(Features);
    flC = KLTCreateFeatureList(Features);

//  Tiles
    TileR:MImage(TileWidth, TileHeight);
    TileC:MImage(TileWidth, TileHeight);

    R:Matrix<double>(2, (XTiles * YTiles));
    C:Matrix<double>(2, (XTiles * YTiles));
    X:Matrix<double>(2, 4);    //  x' = a0 + a1 * x + a2 * y + a3 * x * y
                               //  y' = b0 + b1 * x + b2 * y + b3 * x * y
//  Counters
    i, j:int;
    xt, yt:int;

//  Create regions of interest
    ImageR.CreateROI(0, 0, TileWidth, TileHeight);
    ImageC.CreateROI(0, 0, TileWidth, TileHeight);

//  Rows
    for(yt = 0; yt < YTiles; yt++)
    //  Cols
        for(xt = 0; xt < XTiles; xt++)
        {
        //  Polygon of interest (POI) ?
            if(Polygon != NULL)
            {
            //  Tile inside of POI ?
                if(Polygon->RectInPolygon(
                   (xt * TileWidth), (yt * TileHeight), TileWidth, TileHeight)
```

```
                    == false)
                {
                    R(0, (yt * XTiles + xt)) = -2.0;
                    R(1, (yt * XTiles + xt)) = -2.0;
                    C(0, (yt * XTiles + xt)) = -2.0;
                    C(1, (yt * XTiles + xt)) = -2.0;

                    continue;   //  Next tile
                }
            }

            R(0, (yt * XTiles + xt)) = -1.0;
            R(1, (yt * XTiles + xt)) = -1.0;
            C(0, (yt * XTiles + xt)) = -1.0;
            C(1, (yt * XTiles + xt)) = -1.0;

        //  Set regions of interest
            ImageR.SetROI((xt * TileWidth), (yt * TileHeight), TileWidth, TileHeight);
            ImageC.SetROI((xt * TileWidth), (yt * TileHeight), TileWidth, TileHeight);

        //  Tile from reference image
            TileR.Copy(ImageR);

        //  Tile from image to correct
            TileC.Copy(ImageC);

        //  Select the 25 best features
            KLTSelectGoodFeatures(tc, TileR.GetImageData(), TileWidth, TileHeight, flC);

            for(i = 0; i < Features; i++)
            {
                flR->feature[i]->x = flC->feature[i]->x;
                flR->feature[i]->y = flC->feature[i]->y;
                flR->feature[i]->val = flC->feature[i]->val;
            }

        //  Track the 25 best features
            KLTTrackFeatures(tc, TileR.GetImageData(), TileC.GetImageData(),
                TileWidth, TileHeight, flC);

        //  Search the best tracked feature
            i = 0;
            do
            {
                if((flC->feature[i]->x > -1) && (flC->feature[i]->y > -1))
                {
                //  Reference image
                    R(0, (yt * XTiles + xt)) = (flR->feature[i]->x + (xt * TileWidth));
                    R(1, (yt * XTiles + xt)) = (flR->feature[i]->y + (yt * TileHeight));

                //  Image to correct
                    C(0, (yt * XTiles + xt)) = (flC->feature[i]->x + (xt * TileWidth));
                    C(1, (yt * XTiles + xt)) = (flC->feature[i]->y + (yt * TileHeight));

                    i = Features;
                }
                i++;
            }
            while(i < Features);
        }

//  Delete regions of interest
```

```
    ImageR.DeleteROI();
    ImageC.DeleteROI();

//  Free feature lists
    KLTFreeFeatureList(flR);
    KLTFreeFeatureList(flC);

//  Free tracking context
    KLTFreeTrackingContext(tc);

//  Rows
    for(yt = (YTiles - 1); yt > -1; yt--)
    //  Cols
        for(xt = (XTiles - 1); xt > -1; xt--)
            if( (R(0, (yt * XTiles + xt)) < 0.0) && (R(1, (yt * XTiles + xt)) < 0.0) &&
                (C(0, (yt * XTiles + xt)) < 0.0) && (C(1, (yt * XTiles + xt)) < 0.0))
            {
                R.RemoveCol((yt * XTiles + xt));
                C.RemoveCol((yt * XTiles + xt));
            }

    Point:int;
    do
    {
        X = ComputeCoefficients(R, C);
        Point = ComputeResiduals(R, C, X);

    //  Remove candidate point
        if(Point > -1)
        {
            R.RemoveCol(Point);
            C.RemoveCol(Point);
        }
    }
    while(Point > -1);

//  Warp bilinear function
    return ImageC.WarpBilinear(X);
}
```

– RadiometricCorrection(Image,Image):Image

```
/*
//////////////////////////////////////////////////////////////////////
// Name:        RadiometricCorrection
// Purpose:     Performs the radiometric correction
// Context:     i42aresv Soft
// Returns:     Image
// Parameters:
        ImageR:Image - Reference image
        ImageC:Image - Image to correct
// Notes:
*/
Kernel::RadiometricCorrection(ImageR:Image, ImageC:Image):Image
{
//  Grey levels
    Levels:int = Image::Levels;

//  Equalize histogram for image to correct
    HistogramC:int[Levels];
```

```
    HistogramC = ImageC.EqualizeHistogram();

//  Compute histogram for reference image
    HistogramR:int[Levels];
    HistogramR = ImageR.ComputeHistogram();

//  Transformation functions
    T:int[Levels];
    G:int[Levels];

//  Counters
    i:int;
    j:int;

//  Compute G and T functions
    T[0] = HistogramC[0];
    G[0] = HistogramR[0];
    for(i = 1; i < Levels; i++)
    {
        T[i] = T[i - 1] + HistogramC[i];
        G[i] = G[i - 1] + HistogramR[i];
    }

//  Specified histogram for image to correct
    InvG:int[Levels];

//  Compute Inv(G) function
    MinValue:int;

    for(i = 0; i < Levels; i++)
    {
        MinValue = abs(T[i] - G[0]);
        InvG[i] = 0;
        for(j = 1; j < Levels; j++)
        {
            if(abs(T[i] - G[j]) < MinValue)
            {
                MinValue = abs(T[i] - G[j]);
                InvG[i] = j;
            }
        }
    }

//  Histogram specification
    return ImageC.ContrastStretch(InvG);
}
```

– `ChangesDetection(Image,Image):Image`

```
/*
//////////////////////////////////////////////////////////////////////
// Name:        ChangesDetection
// Purpose:     Detects changes in source images
// Context:     i42aresv Soft
// Returns:     Image
// Parameters:
        ImageR:Image - Reference image
        ImageC:Image - Image to correct
// Notes:
*/
```

```
Kernel::ChangesDetection(ImageR:Image, ImageC:Image):Image
{
    switch(Mode)
    {
    case Absolute:
        return ImageC.Subtract_Abs(ImageR);
        break;
    case Offset:
        return ImageC.Subtract_Offset(ImageR, Offset);
        break;
    default:
        cerr << "ERROR <ChangesDetection>: (Mode != Abs) && (Mode != Offset)" << endl;
        exit(-1);
    }
}
```

– ComputeCoefficients(Matrix<double>,Matrix<double>):Matrix<double>

```
/*
//////////////////////////////////////////////////////////////////////
// Name:        ComputeCoefficients
// Purpose:     Computes affine/bilinear transform coefficients
// Context:     i42aresv Soft
// Returns:     Matrix<double>
// Parameters:
        R:Matrix<double> - GCPs coordinates (Reference image)
        C:Matrix<double> - GCPs coordinates (Image to correct)
// Notes:
*/
Kernel::ComputeCoefficients(R:Matrix<double>, C:Matrix<double>):Matrix<double>
{
    i:int; //  Cols
    j:int; //  Rows
    k:int; //  Points

    NPoints:int = R.GetCols();        //  C.GetCols();
    NTerms:int = 4;   //  x' = a0 + a1 * x + a2 * y + a3 * x * y
                      //  y' = b0 + b1 * x + b2 * y + b3 * x * y

    D:Matrix<double>(NTerms, NPoints);  //  Data matrix
    V:Matrix<double>(NTerms, NTerms);   //  Variances matrix (square)

    X:Matrix<double>(2, NTerms); //  Coefficients matrix

    M:double[NTerms];    //  Means vector

//  Data
    for(k = 0; k < NPoints; k++)
    {
        D(1, k) = C(0, k);  //  x
        D(2, k) = C(1, k);  //  y
        D(3, k) = (C(0, k) * C(1, k));  //  x * y
    }

//  Means
    for(j = 1; j < NTerms; j++)
    {
        M[j] = 0.0;
        for(k = 0; k < NPoints; k++)
            M[j] += D(j, k);
```

```
        M[j] /= NPoints;
    }

//  Variances
    for(j = 1; j < NTerms; j++)
        for(i = 1; i < NTerms; i++)
        {
            V(j, i) = 0.0;
            for(k = 0; k < NPoints; k++)
                V(j, i) += ((D(j, k) - M[j]) * (D(i, k) - M[i]));
            V(j, i) /= NPoints;
        }

    l:int;
    for(l = 0; l < 2; l++)
    {
    //  Data
        for(k = 0; k < NPoints; k++)
            D(0, k) = R(l, k);

    //  Means
        M[0] = 0.0;
        for(k = 0; k < NPoints; k++)
            M[0] += D(0, k);
        M[0] /= NPoints;

    //  Variances
        V(0, 0) = 0.0;
        for(k = 0; k < NPoints; k++)
            V(0, 0) += ((D(0, k) - M[0]) * (D(0, k) - M[0]));
        V(0, 0) /= NPoints;

        for(j = 0; j < NTerms; j++)
        {
            V(j, 0) = 0.0;
            for(k = 0; k < NPoints; k++)
                V(j, 0) += ((D(j, k) - M[j]) * (D(0, k) - M[0]));
            V(j, 0) /= NPoints;
        }

        for(i = 0; i < NTerms; i++)
        {
            V(0, i) = 0.0;
            for(k = 0; k < NPoints; k++)
                V(0, i) += ((D(0, k) - M[0]) * (D(i, k) - M[i]));
            V(0, i) /= NPoints;
        }

    //  Coefficients (a1, aN)
        for(i = 1; i < NTerms; i++)
            X(l, i) = -(pow(-1, i) * V.SubMatrix(0, i).Determinant() /
                            V.SubMatrix(0, 0).Determinant());

    //  Coefficient a0
        X(l, 0) = M[0];
        for(i = 1; i < NTerms; i++)
            X(l, 0) -= (X(l, i) * M[i]);
    }

//  Coefficients
    return X;
}
```

– ComputeResiduals(Matrix<double>,Matrix<double>,Matrix<double>):int

```
/*
//////////////////////////////////////////////////////////////////////
// Name:        ComputeResiduals
// Purpose:     Computes GCPs residuals and selects the worst point
// Context:     i42aresv Soft
// Returns:     int
// Parameters:
        R:Matrix<double> - GCPs coordinates (Reference image)
        C:Matrix<double> - GCPs coordinates (Image to correct)
        X:Matrix<double> - Coefficients matrix
// Notes:
*/
Kernel::ComputeResiduals(R:Matrix<double>, C:Matrix<double>, X:Matrix<double>):int
{
    NPoints:int = R.GetCols(); //  C.GetCols();

//  Values
    x, y:double;

    EL:double;  //  Residual
    MaxEL:double = 0.0;  //  Residual with max value (candidate point)

    RMS:double = 0.0;  //  Root Mean Squared

    Point:int = -1;  //  Candidate point

    k:int;
    for(k = 0; k < NPoints; k++)
    {
        x = X(0, 0) + (X(0, 1) * C(0, k)) +
            (X(0, 2) * C(1, k)) + (X(0, 3) * C(0, k) * C(1, k));
        y = X(1, 0) + (X(1, 1) * C(0, k)) +
            (X(1, 2) * C(1, k)) + (X(1, 3) * C(0, k) * C(1, k));

        EL = sqrt((pow((R(0, k) - x), 2) + pow((R(1, k) - y), 2)));

    //  Select the worst point (candidate point)
        if((EL > 1.0) && (EL > MaxEL))
        {
            MaxEl = EL;
            Point = k;
        }

        RMS += (pow((R(0, k) - x), 2) + pow((R(1, k) - y), 2));
    }

//  Compute root mean squared
    RMS /= NPoints;

    if(sqrt(RMS) > 1.0)
        return Point;    //  The worst point
    else
        return -1;
}
```

Clase Image

– WarpBilinear(double[][]):Image

```
/*
//////////////////////////////////////////////////////////////////////
// Name:        WarpBilinear
// Purpose:     Performs a bilinear transform with the specified coefficients
// Context:     i42aresv Soft
// Returns:
// Parameters:
        X:Matrix<double> - Bilinear transform coefficients
// Notes:
*/
Image::WarpBilinear(X:Matrix<double>)
{
//  Counters
    i:int;
    j:int;

//  Coefficients
    dCoeffs:double[2][4];

//  Prepare array to iplWarpBilinear
    for(j = 0; j < 2; j++)
    {
        dCoeffs[j][3] = X(j, 0);
        for(i = 1; i < 3; i++)
            dCoeffs[j][i] = X(j, i);
        dCoeffs[j][0] = X(j, 3);
    }

//  Warp bilinear
    iplWarpBilinear(Image, Image, dCoeffs,
        IPL_WARP_R_TO_Q, IPL_INTER_CUBIC | IPL_SMOOTH_EDGE);
}
```

– ComputeHistogram():int[]

```
/*
//////////////////////////////////////////////////////////////////////
// Name:        ComputeHistogram
// Purpose:     Computes the intensity histogram
// Context:     i42aresv Soft
// Returns:     int[]
// Parameters:
// Notes:
*/
Image::ComputeHistogram():int[]
{
//  Histogram
    Histogram:int[Levels];

//  Number of tiles
    XTiles:int = (GetWidth() / TileWidth);  //  Horizontal
    YTiles:int = (GetHeight() / TileHeight);    //  Vertical

//  DN
    Pixel:byte;

//  Counters
    xt, yt:int;    //  Tile
    xp, yp:int;    //  Pixel
```

```
//  Tile
    Tile:MImage(TileWidth, TileHeight);

//  Clear histogram
    memset(Histogram, 0, (Levels * sizeof(int)));

//  Create region of interest
    CreateROI(0, 0, TileWidth, TileHeight);

//  Rows
    for(yt = 0; yt < YTiles; yt++)
    //  Cols
        for(xt = 0; xt < XTiles; xt++)
        {
        //  Set region of interest
            SetROI((xt * TileWidth), (yt * TileHeight), TileWidth, TileHeight);

        //  Tile from image
            Tile.Copy(this);

        //  Rows
            for(yp = 0; yp < TileHeight; yp++)
            //  Cols
                for(xp = 0; xp < TileWidth; xp++)
                {
                    Pixel = Tile.GetPixel(xp, yp);
                    Histogram[Pixel]++;
                }
        }

//  Delete region of interest
    DeleteROI();

    return Histogram;
}
```

– `ContrastStretch(int[])`

```
/*
//////////////////////////////////////////////////////////////////////
// Name:        ContrastStretch
// Purpose:     Stretches the contrast of an image using intensity transformation
// Context:     i42aresv Soft
// Returns:
// Parameters:
        Histogram:int - Intensity transformation
// Notes:
*/
Image::ContrastStretch(Histogram:int[])
{
//  Number of tiles
    XTiles:int = (GetWidth() / TileWidth);  //  Horizontal
    YTiles:int = (GetHeight() / TileHeight);    //  Vertical

//  DN
    Pixel:byte;

//  Counters
    xt, yt:int;    //  Tile
    xp, yp:int;    //  Pixel
```

```
//  Tile
    Tile:MImage(TileWidth, TileHeight);

//  Create region of interest
    CreateROI(0, 0, TileWidth, TileHeight);

//  Rows
    for(yt = 0; yt < YTiles; yt++)
    //  Cols
        for(xt = 0; xt < XTiles; xt++)
        {
        //  Set region of interest
            SetROI((xt * TileWidth), (yt * TileHeight), TileWidth, TileHeight);

        //  Tile from image
            Tile.Copy(this);

        //  Rows
            for(yp = 0; yp < TileHeight; yp++)
            //  Cols
                for(xp = 0; xp < TileWidth; xp++)
                {
                    Pixel = Tile.GetPixel(xp, yp);
                    Pixel = Histogram[Pixel];
                    Tile.PutPixel(xp, yp, Pixel);
                }

        //  Save tile
            Copy(Tile);
        }

//  Delete region of interest
    DeleteROI();
}
```

– EqualizeHistogram():Image

```
/*
////////////////////////////////////////////////////////////////////
// Name:        EqualizeHistogram
// Purpose:     Enhances an image by flattening its intensity histogram
// Context:     i42aresv Soft
// Returns:     int[]
// Parameters:
// Notes:
*/
void Image::EqualizeHistogram():int[]
{
//  Histogram
    Histogram:int[Levels];

//  F function
    F:double[Levels];

//  T function (transformation function)
    T:int[Levels];

//  Number of tiles
    XTiles:int = (GetWidth() / TileWidth);  //  Horizontal
```

```
    YTiles:int = (GetHeight() / TileHeight);    //  Vertical

//  DN
    Pixel:byte;

//  Counters
    i:int;

    xt, yt:int;    //  Tile
    xp, yp:int;    //  Pixel

//  Tile
    Tile:MImage(TileWidth, TileHeight);

//  Clear histogram
    memset(Histogram, 0, (Levels * sizeof(int)));

//  Compute histogram
    Histogram = ComputeHistogram();

//  Factor
    Factor:double = (1.0 / (GetWidth() * GetHeight()));

//  Compute F and T function
    F[0] = (Histogram[0] * Factor);
    T[0] = (int)(F[0] * (Levels - 1) + 0.5);
    for(i = 1; i < Levels; i++)
    {
        F[i] = F[(i - 1)] + (Histogram[i] * Factor);
        T[i] = (int)F[i] * (Levels - 1) + 0.5);
    }

//  Clear histogram
    memset(Histogram, 0, (Levels * sizeof(int)));

//  Create regions of interest
    CreateROI(0, 0, TileWidth, TileHeight);

//  Rows
    for(yt = 0; yt < YTiles; yt++)
    //  Cols
        for(xt = 0; xt < XTiles; xt++)
        {
        //  Set regions of interest
            SetROI((xt * TileWidth), (yt * TileHeight), TileWidth, TileHeight);

        //  Tile from image
            Tile.Copy(this);

        //  Rows
            for(yp = 0; yp < TileHeight; yp++)
            //  Cols
                for(xp = 0; xp < TileWidth; xp++)
                {
                    Pixel = Tile.GetPixel(xp, yp);
                    Pixel = T[Pixel];
                    Tile.PutPixel(xp, yp, Pixel);

                //  Histogram
                    pHistogram[Pixel]++;
                }
```

```
        //  Save tile
            Copy(Tile);
        }

//  Delete region of interest
    DeleteROI();

    return Histogram;
}
```

– Subtract_Abs(Image):Image

```
/*
//////////////////////////////////////////////////////////////////////
// Name:        Subtract_Abs
// Purpose:     Subtracts pixel values of one image from those of another
                image and computes absolute pixel values of resultant image
// Context:     i42aresv Soft
// Returns:     Image
// Parameters:
        ImageC:Image - Image to correct
// Notes:
*/
Image::Subtract_Abs(ImageC:Image):Image
{
//  Number of tiles
    XTiles = (GetWidth() / TileWidth);  //  Horizontal
    YTiles = (GetHeight() / TileHeight);    //  Vertical

//  DN
    Pixel:byte;
    PixelC:byte;

//  Counters
    xt, yt:int;    //  Tile
    xp, yp:int;    //  Pixel

//  Tiles
    Tile:MImage(TileWidth, TileHeight);  //  this
    TileC:MImage(TileWidth, TileHeight);

//  Create regions of interest
    CreateROI(0, 0, TileWidth, TileHeight);
    ImageC.CreateROI(0, 0, TileWidth, TileHeight);

//  Rows
    for(yt = 0; yt < YTiles; yt++)
    //  Cols
        for(xt = 0; xt < XTiles; xt++)
        {
        //  Set regions of interest
            SetROI((xt * TileWidth), (yt * TileHeight), TileWidth, TileHeight);
            ImageC.SetROI((xt * TileWidth), (yt * TileHeight), TileWidth, TileHeight);

        //  Tile from reference image
            Tile.Copy(this);

        //  Tile from image to correct
            TileC.Copy(ImageC);
```

```
        //  Rows
            for(yp = 0; yp < TileHeight; yp++)
            //  Cols
                for(xp = 0; xp < TileWidth; xp++)
                {
                    Pixel = Tile.GetPixel(xp, yp);
                    PixelC = TileC.GetPixel(xp, yp);

                    Pixel = abs(Pixel - PixelC);
                    Tile.PutPixel(xp, yp, Pixel);
                }

        //  Set tile
            ImageC.Copy(Tile);
        }

//  Delete regions of interest
    DeleteROI();
    ImageC.DeleteROI();

    return ImageC;
}
```

– Subtract_Offset(Image,int):Image

```
/*
//////////////////////////////////////////////////////////////////////
// Name:        Subtract_Offset
// Purpose:     Subtracts pixel values of one image from those of another
                image and adds a constant to pixel values of the resultant image
// Context:     i42aresv Soft
// Returns:     Image
// Parameters:
        ImageC:Image - Image to correct
        Scalar:byte - Offset
// Notes:
*/
Image::Subtract_Offset(ImageC:Image, Scalar:byte):Image
{
//  Number of tiles
    XTiles:int = (GetWidth() / TileWidth);  //  Horizontal
    YTiles:int = (GetHeight() / TileHeight);    //  Vertical

//  DN
    Pixel:byte;
    PixelC:byte;

//  Counters
    xt, yt:int;    //  Tile
    xp, yp:int;    //  Pixel

//  Tiles
    Tile:MImage(TileWidth, TileHeight);  //  this
    TileC:MImage(TileWidth, TileHeight);

//  Create regions of interest
    CreateROI(0, 0, TileWidth, TileHeight);
    ImageC.CreateROI(0, 0, TileWidth, TileHeight);

//  Rows
```

```
    for(yt = 0; yt < YTiles; yt++)
    //  Cols
        for(xt = 0; xt < XTiles; xt++)
        {
        //  Set regions of interest
            SetROI((xt * TileWidth), (yt * TileHeight), TileWidth, TileHeight);
            ImageC.SetROI((xt * TileWidth), (yt * TileHeight), TileWidth, TileHeight);

        //  Tile from reference image
            Tile.Copy(this);

        //  Tile from image to correct
            TileC.Copy(ImageC);

        //  Rows
            for(yp = 0; yp < TileHeight; yp++)
            //  Cols
                for(xp = 0; xp < TileWidth; xp++)
                {
                    Pixel = Tile.GetPixel(xp, yp);
                    PixelC = TileC.GetPixel(xp, yp);
                    Pixel = esc(
                        0,                          //  DN = 0 (Min.)
                        (Pixel - PixelC + Scalar),
                        (Levels - 1));              //  DN = 255 (Max.)
                    Tile.PutPixel(xp, yp, Pixel);
                }

        //  Set tile
            ImageC.Copy(Tile);
        }

//  Delete regions of interest
    DeleteROI();
    ImageC.DeleteROI();

    return ImageC;
}
```

Clase Matrix<Tipo>

– Determinant():Tipo

```
/*
//////////////////////////////////////////////////////////////////////
// Name:        Determinant
// Purpose:     Computes the matrix determinant
// Context:     i42aresv Soft
// Returns:     Tipo
// Parameters:
// Notes:
*/
Matrix<Tipo>::Determinant():Tipo
{
    if(Rows != Cols)
    {
        cerr << "ERROR <Matrix::Determinant>: Rows != Cols" << endl;
        exit(-1);
    }
```

```
    Determinant:Tipo;  //  Result
    a0i:Tipo;
    A0i:Tipo;

    if((Rows == 1) && (Cols == 1))
        Determinant = Data[0];
    else
    {
        Determinant = (Tipo)0;

        i:int;
        for(i = 0; i < Cols; i ++)
        {
            a0i = Data[i];  //  ai0 element
            A0i = pow(-1, i) * SubMatrix(0, i).Determinant();
            Determinant += a0i * A0i;
        }
    }

//  Matrix determinant
    return Determinant;
}
```

– SubMatrix(int,int):Matrix<Tipo>

```
/*
////////////////////////////////////////////////////////////////////
// Name:        SubMatrix
// Purpose:     Extracts a matrix by removing row Row and column Col
// Context:     i42aresv Soft
// Returns:     Matrix<Tipo>
// Parameters:
        Row:int - Row
        Col:int - Column
// Notes:
*/
Matrix<Tipo>::SubMatrix(Row:int, Col:int):Matrix<Tipo>
{
    SubMatrix:Matrix<Tipo>(this);

    SubMatrix.RemoveRow(Row);
    SubMatrix.RemoveCol(Col);

    return SubMatrix;
};
```

– RemoveRow(int)

```
/*
////////////////////////////////////////////////////////////////////
// Name:        RemoveRow
// Purpose:     Removes row Row
// Context:     i42aresv Soft
// Returns:
// Parameters:
        Row:int - Row
// Notes:
```

```
*/
Matrix<Tipo>::RemoveRow(int Row)
{
    if(Rows < 2)
    {
        cerr << "ERROR <Matrix::RemoveRow>: (Rows < 2)" << endl;
        exit(-1);
    }
    if((Row < 0) || (Row > (Rows - 1)))
    {
        cerr << "ERROR <Matrix::RemoveRow>: (Row < 0) || (Row > (Rows - 1))" << endl;
        exit(-1);
    }

    Size:int = ((Rows - 1) * Cols);

    TmpData:Tipo[Size];

    i:int;
    j:int;
    k:int = 0;
    for(j = 0; j < Rows; j++)
    {
        if(j == Row)
            continue;

        for(i = 0; i < Cols; i++)
        {
            TmpData[k] = Data[j * Cols + i];
            k++;
        }
    }
    Data = TmpData;
    Rows--;
}
```

– RemoveCol(int)

```
/*
//////////////////////////////////////////////////////////////////////
// Name:        RemoveCol
// Purpose:     Removes column Col
// Context:     i42aresv Soft
// Returns:
// Parameters:
        Col:int - Column
// Notes:
*/
Matrix<Tipo>::RemoveCol(Col:int)
{
    if(Cols < 2)
    {
        cerr << "ERROR <Matrix::RemoveCol>: Cols < 2" << endl;
        exit(-1);
    }
    if((Col < 0) || (Col > (Cols - 1)))
    {
        cerr << "ERROR <Matrix::RemoveCol>: (Col < 0) || (Col > (Cols - 1))" << endl;
        exit(-1);
    }
```

```
    Size:int = (Rows * (Cols - 1));

    TmpData:Tipo[Size];

    i:int;
    j:int;
    k:int = 0;
    for(j = 0; j < Rows; j++)
        for(i = 0; i < Cols; i++)
        {
            if(i == Col)
                continue;

            TmpData[k] = Data[j * Cols + i];
            k++;
        }
    Data = TmpData;
    Cols--;
}
```

Clase Polygon

– PointInPolygon(int,int):bool

```
/*
//////////////////////////////////////////////////////////////////////
// Name:        PointInPolygon
// Purpose:     Checks whether the point specified is inside of the boundaries
                of the polygon.
// Context:     i42aresv Soft
// Returns:     bool
// Parameters:
        x, y:int - Coordinates of point
// Notes:
*/
Polygon::PointInPolygon(x:int, y:int):bool
{
//  Controls
    LeftOfPoint:int = 0;
    RightOfPoint:int = 0;

//  X coordinate of the intersection
    XIntersection:double;

//  Counter
    i:int;

//  Closed polygon
    for(i = 1; i < Size; i++)
    {
        if( ((Polygon[(i - 1)].y <= y) && (Polygon[i].y > y)) ||
            ((Polygon[(i - 1)].y > y) && (Polygon[i].y <= y)))
        {
            XIntersection = Polygon[(i - 1)].x +
                ((y - Polygon[(i - 1)].y) *
                    (Polygon[i].x - Polygon[(i - 1)].x) /
                    (Polygon[i].y - Polygon[(i - 1)].y));
            if(XIntersection < x)
```

```
                LeftOfPoint++;
            if(XIntersection > x)
                RightOfPoint++;
        }
    }
    if((LeftOfPoint % 2 == 1) && (RightOfPoint % 2 == 1))
        return true;
    else
        return false;
}
```

– `RectInPolygon(int,int,int,int):bool`

```
/*
//////////////////////////////////////////////////////////////////////
// Name:        RectInPolygon
// Purpose:     Determines whether any part of the rectangle specified is
                inside of the boundaries of the polygon.
// Context:     i42aresv Soft
// Returns:     bool
// Parameters:
        x, y:int - Offsets from the origin of the rectangular region
        Width, Height:int - Size of rectangular region
// Notes:
*/
Polygon::RectInPolygon(x:int, y:int, Width:int, Height:int):bool
{
    if( (PointInPolygon(x, y) == true) ||
        (PointInPolygon(x, (y + Height - 1)) == true) ||
        (PointInPolygon((x + Width - 1), y) == true) ||
        (PointInPolygon((x + Width - 1), (y + Height - 1)) == true))
        return true;
    else
        return false;
}
```

Como comentamos anteriormente, estas son las operaciones más relevantes de las clases descritas en el modelo de objetos, para una relación completa de las clases, atributos y operaciones, ya codificadas, ver el apéndice A.

Capítulo 8

Implementación

8.1 Introducción

En el capítulo anterior se ha analizado y diseñado el núcleo del sistema software, las clases (atributos y operaciones) y la relación con los distintos componentes software (sección 7.3). En este capítulo se atenderán aspectos de implementación: módulos, interfaz gráfico, ajuste del algoritmo de búsqueda y seguimiento de características, etc.

8.2 Módulos e interfaz gráfico

Inicialmente, se contempló la posibilidad de realizar un sólo módulo, encargado de realizar todas y cada una de las tareas del proceso de detección de cambios. Posteriormente, por motivos de flexibilidad y facilidad para incorporarlos como etapas independientes en paquetes GIS comerciales (GRASS), se consideró la posibilidad de unir a este módulo inicial, encargado de realizar todo el proceso de detección de cambios, tres módulos independientes, encargados de realizar las tareas de corrección geométrica, radiométrica y diferencia de imágenes (con procesamiento añadido) respectivamente. Y realizar un pequeño *front end* con *scripts* TCL/TK que permitiesen gestionar fácilmente los distintos módulos.

Naturalmente, todos los módulos se pueden ejecutar desde la línea de comando, lo que facilita notablemente su incorporación a paquetes ya existentes en el mercado.

8.2.1 Módulo DetCam

El módulo `DetCam` (ver figura 8.1) es el encargado de realizar el *proceso de detección de cambios* de forma completa. `DetCam` registra información adicional para usos posteriores (marcado de GCPs, etc.). Alguna de la información suministrada por el programa `DetCam` es la siguiente:

1. GCPs detectados y seguidos
2. GCPs eliminados por producir desajustes
3. Tiles fuera del polígono de interés
4. Coeficientes del polinomio de ajuste geométrico
5. Función inversa (G^{-1}) de ajuste radiométrico
6. Tipo de procesamiento de la imagen diferencia

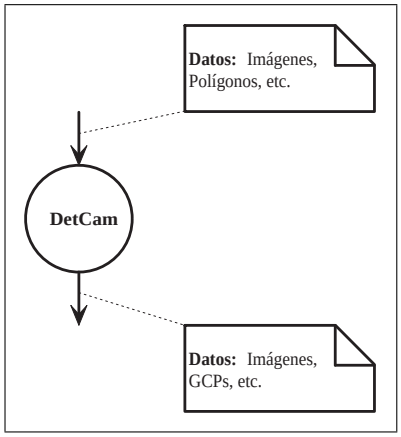


Figura 8.1: Módulo DetCam.

Script TCL/TK

En la figura 8.2 podemos observar el interfaz diseñado para ejecutar el módulo `DetCam`.

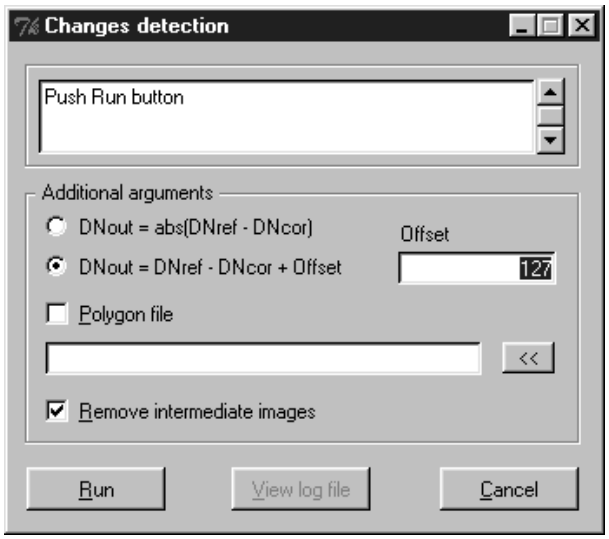


Figura 8.2: Detección de cambios.

8.2.2 Módulo CorGeo

El módulo `CorGeo` (ver figura 8.3) es el encargado de realizar la etapa de *corrección geométrica*.

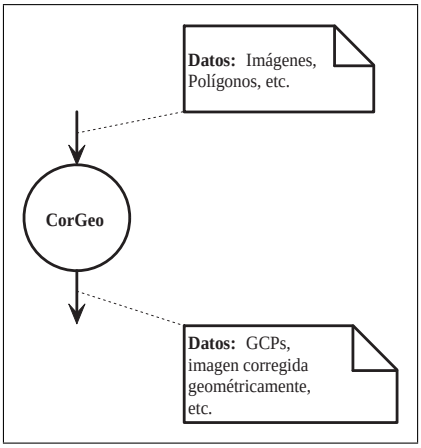


Figura 8.3: Módulo CorGeo.

Script TCL/TK

En la figura 8.4 podemos observar el interfaz diseñado para ejecutar el módulo `CorGeo`.

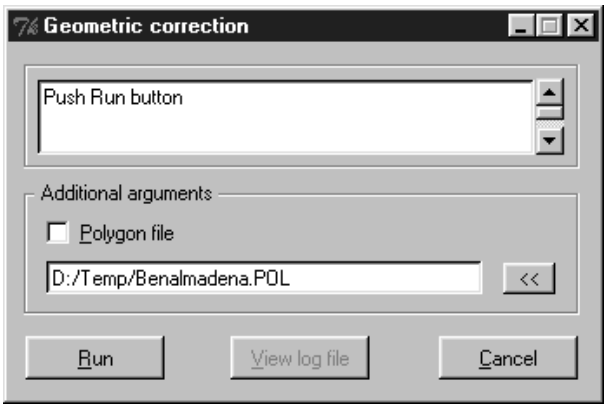


Figura 8.4: Corrección geométrica.

8.2.3 Módulo CorRad

El módulo `CorRad` (ver figura 8.5) es el encargado de realizar la etapa de *corrección radiométrica*.

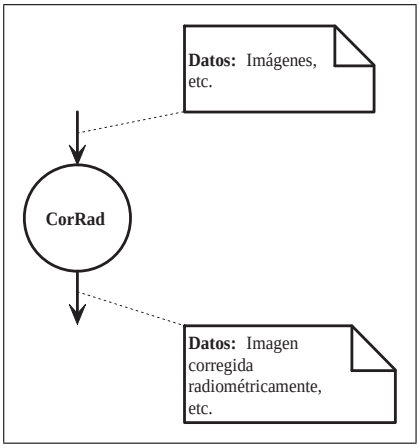


Figura 8.5: Módulo CorRad.

Script TCL/TK

En la figura 8.6 podemos observar el interfaz diseñado para ejecutar el módulo `CorRad`.

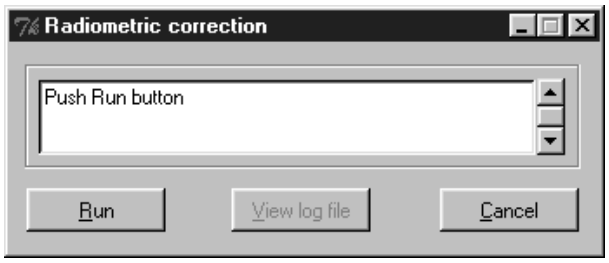


Figura 8.6: Corrección radiométrica.

8.2.4 Módulo DifIma

El módulo `DifIma` (ver figura 8.7) es el encargado de realizar la *diferencia de imágenes* y su posterior procesamiento (valor absoluto o desplazamiento).

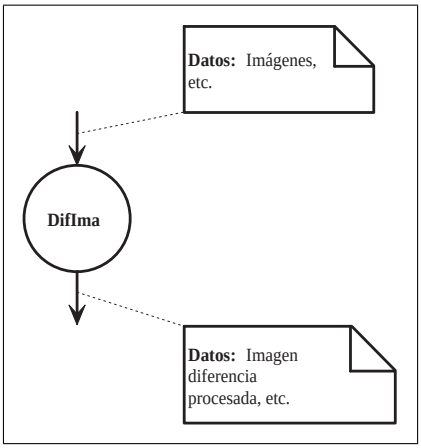


Figura 8.7: Módulo DifIma.

Script TCL/TK

En la figura 8.8 podemos observar el interfaz diseñado para ejecutar el módulo `DifIma`.

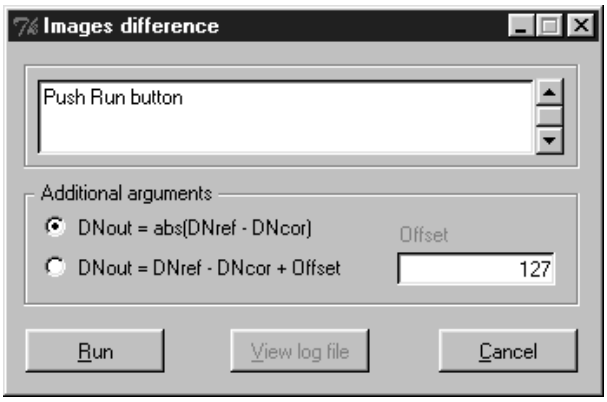


Figura 8.8: Diferencia de imágenes.

En resumen, el proceso de detección de cambios se puede abordar de dos formas: 1) ejecutando directamente el módulo `DetCam` con sus correspondientes parámetros o 2) ejecutando secuencialmente los módulos `CorGeo`, `CorRad` y `DifIma` con sus correspondientes parámetros, utilizando la imagen de salida de cada módulo como parámetro en la llamada del siguiente módulo (ver figura 8.9).

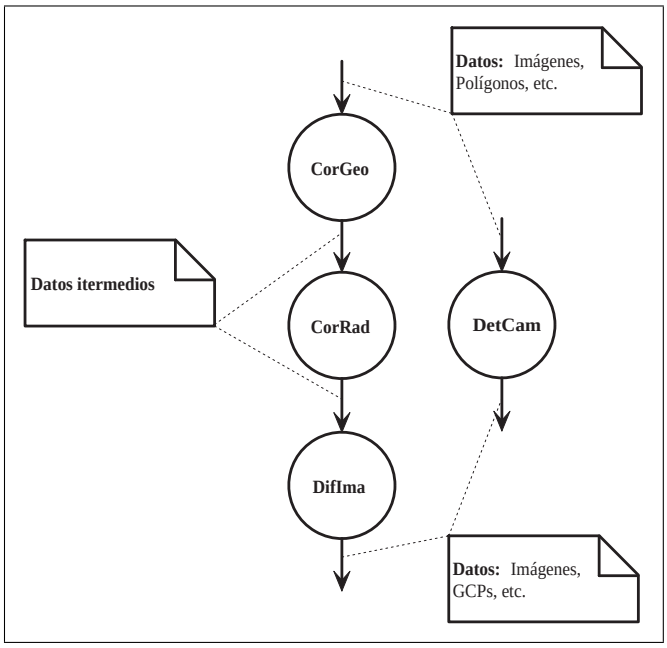


Figura 8.9: Módulos.

Los módulos `CorGeo`, `CorRad` y `DifIma` registran, al igual que el módulo `DetCam`, información adicional relativa a cada una de las etapas que realizan. Si reuniésemos la información suministrada por estos módulos en un mismo fichero coincidiría con la proporcionada por el módulo `DetCam`.

Script TCL/TK

En la figura 8.10 podemos observar el interfaz diseñado para establecer los parámetros comunes a cada uno de los módulos, y gestionar la llamada de cada uno de ellos.

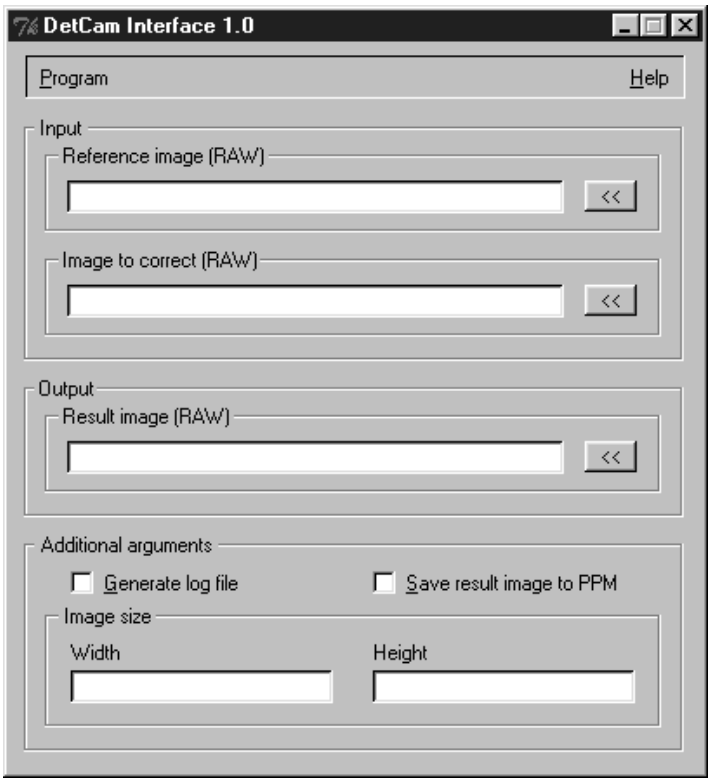


Figura 8.10: Interfaz de usuario.

NOTA: En el apéndice A podemos ver todos y cada uno de los *scripts* utilizados en el desarrollo del interfaz.

8.3 Ajuste del tracker KLT

En la operación `CKernel::GeometricCorrection(Image,Image):Image` especificada en la sección 7.5 hacíamos referencia a algunas de las funciones y estructuras proporcionadas por el *KLT Feature Tracker 1.1.5*, concretamente a las operaciones:

- `void KLTSetVerbosity(int verbosity);`

  Activa o desactiva el modo *verbose*

- `KLT_TrackingContext KLTCreateTrackingContext(void);`

  Crea e inicializa una estructura `KLT_TrackingContext`

  `void KLTFreeTrackingContext(KLT_TrackingContext tc);`

  Libera una estructura `KLT_TrackingContext`

- `KLT_FeatureList KLTCreateFeatureList(int nFeatures);`

  Crea e inicializa una estructura `KLT_FeatureList`

  `void KLTFreeFeatureList(KLT_FeatureList fl);`

  Libera una estructura `KLT_FeatureList`

- `void KLTUpdateTCBorder(KLT_TrackingContext tc);`

  Calcula y actualiza los campos `borderx` y `bordery` de la estructura `tc`

- `void KLTSelectGoodFeatures(KLT_TrackingContext tc,`
  `KLT_PixelType *img,`
  `int ncols, int nrows, KLT_FeatureList fl);`

  Selecciona características en la imagen apuntada por `img`[1] y las almacena en `fl`

- `void KLTTrackFeatures(KLT_TrackingContext tc,`
  `KLT_PixelType *img1, KLT_PixelType *img2,`
  `int ncols, int nrows, KLT_FeatureList fl);`

  Localiza características en la imagen apuntada por `img2` que coinciden

[1] KLT_PixelType es por defecto `unsigned char`.

con las seleccionadas (seguimiento) en la imagen apuntada por `img2` y las almacena en `fl`

y a las estructuras:

- ```
  typedef struct {
  int mindist;
  int window_width, window_height;
  KLT_BOOL sequentialMode;
  KLT_BOOL smoothBeforeSelecting;
  KLT_BOOL writeInternalImages;
  int min_eigenvalue;
  float min_determinant;
  float min_displacement;
  int max_iterations;
  float max_residue;
  float grad_sigma;
  float smooth_sigma_fact;
  float pyramid_sigma_fact;
  int nSkippedPixels;
  int borderx;
  int bordery;
  int nPyramidLevels;
  int subsampling;
  void *pyramid_last;
  void *pyramid_last_gradx;
  void *pyramid_last_grady;
  } *KLT_TrackingContext;
  ```

  `KLT_TrackingContext` contiene los parámetros que configuran el motor de búsqueda y seguimiento de características. Son muchos los campos que de algún modo influyen en el comportamiento del motor. Entre todos ellos destacan por su importancia, los siguientes:

  1. `int window_width, window_height;` Dimensiones de la ventana de características (sí modificamos las dimensiones de la ven-
  ```

tana de características es imprescindible ajustar `int borderx, bordery;`)

2. `int borderx, bordery;` Dimensiones del borde, en píxeles, que no es analizado en la búsqueda de características.

   NOTA: Este borde es necesario ya que la convolución con la *Gaussina* provoca que muchos de los valores de la imagen sean desconocidos. El seguimiento de características en estas regiones puede producir resultados extraños.

Alguno de los parámetros se pueden ajustar para reducir el tiempo de computación, generar imágenes intermedias, etc.

Para una descripción detallada de cada uno de estos parámetros ver `http://vision.stanford.edu/~birch/klt/`.

- ```
  typedef struct {
  int nFeatures;
  KLT_Feature *feature;
  } *KLT_FeatureList;
  ```

  `KLT_FeatureList` es una lista de características. El campo `int nFeatures` indica el número de características a localizar (número máximo).

- ```
  typedef struct {
  KLT_locType x;
  KLT_locType y;
  int val;
  } *KLT_Feature;
  ```

  `KLT_Feature` es una característica. Los campos `KLT_locType x, y;` indican su localización y el campo `int val` su valor. El valor de una característica se interpreta de la siguiente forma:

  1. `KLT_TRACKED (0)`

     Característica localizada y seguida

  2. `KLT_NOT_FOUND (-1)`

     Característica no localizada

3. `KLT_SMALL_DET (-2)`

   Característica no considerada. El determinante de la matriz gradiente es inferior al especificado (`float min_determinant;`)

4. `KLT_MAX_ITERATIONS (-3)`

   Característica no considerada. El número de iteraciones es superior al especificado (`int max_iterations;`)

5. `KLT_OOB (-4)`

   Característica no considerada. La característica está fuera del límite (`int borderx, bordery;`)

6. `KLT_LARGE_RESIDUE (-5)`

   Característica no considerada. El residuo entre dos características es superior al especificado (`float max_residue;`)

Parte III

Pruebas y resultados

Capítulo 9

Pruebas

9.1 Introducción

Este capítulo presenta las pruebas realizadas a lo largo del proyecto para calibrar (o medir) la bondad de las técnicas empleadas y su posterior implementación. Las pruebas se presentarán atendiendo, como hemos venido haciendo en capítulos precedentes, a las etapas del proceso de detección de cambios.

Comenzaremos con la batería de imágenes sintéticas utilizadas en el ajuste de la etapa de corrección geométrica, a continuación, las imágenes de satélite utilizadas (regiones características) tanto en la etapa de corrección geométrica como radiométrica. Finalizaremos con una descripción detallada de las pruebas realizadas y el objeto de las mismas, dejando para el capítulo posterior la presentación de los resultados obtenidos.

9.2 Imágenes

Con objeto de comprobar si las distintas técnicas implementadas logran el objetivo para el que han sido diseñadas, hemos utilizado dos tipos de imágenes:

1. Imágenes sintéticas (256×256 píxeles)

2. Imágenes de satélite (512×512 píxeles)

Las primeras diseñadas para utilizarlas fundamentalmente en la etapa de corrección geométrica y las segundas extraídas de las imágenes proporcionadas por el proveedor, para utilizarlas en todas las etapas del proceso de detección de cambios.

9.2.1 Imágenes sintéticas

Las imágenes sintéticas te permiten obtener resultados que resultan difícilmente apreciables con las imágenes de satélite convencionales. El diseño de imágenes sintéticas es crucial en cualquier proceso de calibrado y prueba de técnicas de procesamiento de imágenes que se precie y como tal, hemos diseñado una batería de ellas para medir la calidad del motor de búsqueda, seguimiento y eliminación de GCPs.

1. Traslación (10 píxeles $\downarrow$ y 10 píxeles $\rightarrow$)
2. Escalado (10 píxeles $\rightarrow\leftarrow$ y 10 píxeles $\begin{smallmatrix}\downarrow\\\uparrow\end{smallmatrix}$)
3. Inclinación (15 píxeles $\overset{\rightarrow}{}$ y 15 píxeles $\underset{\leftarrow}{}$)
4. Perspectiva (10 píxeles $\overset{\rightarrow\leftarrow}{}$ y 10 píxeles $\underset{\leftarrow\rightarrow}{}$)
5. Rotación (5 grados $\curvearrowright$)

Imágenes

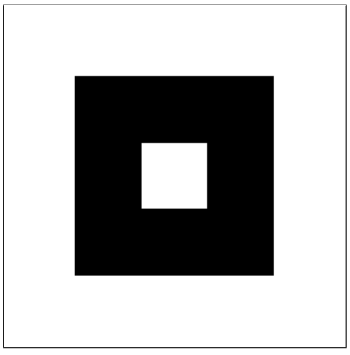


Figura 9.1: Original.

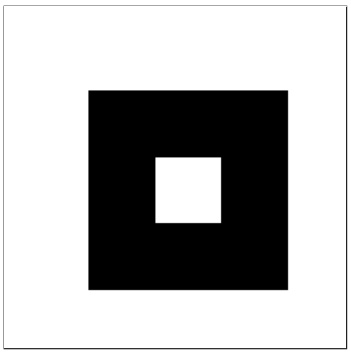


Figura 9.2: Traslación.

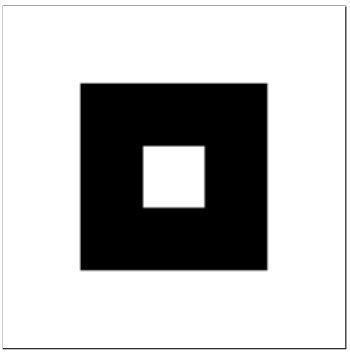


Figura 9.3: Escalado.

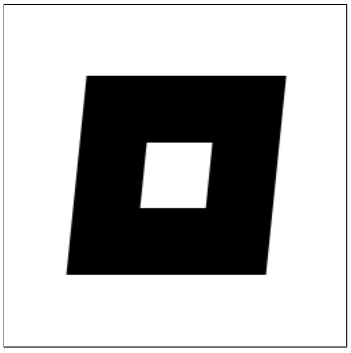


Figura 9.4: Inclinación.

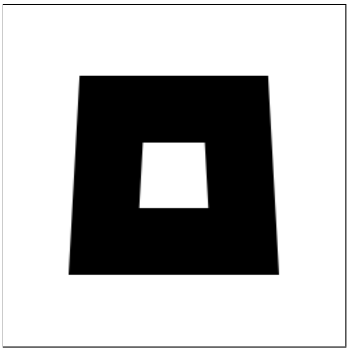


Figura 9.5: Perspectiva.

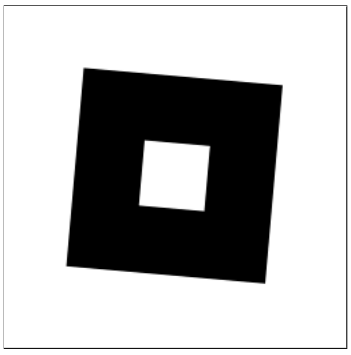


Figura 9.6: Rotación.

9.2.2 Imágenes de satélite

Las imágenes sintéticas son fundamentales en el proceso de calibrado y prueba de la etapa de corrección geométrica, sin embargo distan mucho de las imágenes con las que se trabajan habitualmente. Por tanto, para continuar el proceso de prueba hemos utilizado dos series de imágenes correspondientes a fragmentos del término municipal de Málaga y Benalmádena respectivamente, con una resolución temporal de 6 meses.

Estas series de imágenes serán utilizadas en las etapas de corrección radiométrica y diferencia de imágenes con procesamiento, además de la etapa de corrección geométrica.

Con ambas series hemos tratado de ilustrar la enorme variabilidad espacial del fenómeno urbano, donde se suceden una enorme cantidad de elementos espaciales sin orden ni concierto:

1. La costa
2. Montañas
3. Valles
4. Focos urbanos

Este aspecto es importantísimo, ya que pone de relieve la potencia de la técnica empleada para seleccionar y seguir puntos en imágenes, con independencia del contenido.

Imágenes

Málaga

Hemos considerado apropiado incluir esta serie por que en ella se alternan costa, focos urbanos y grandes espacios deshabitados.

Figura 9.7: Málaga'99.

Figura 9.8: Málaga'00.

Benalmádena

Esta otra serie se caracteriza por la presencia de zonas montañosas y zonas habitadas, este tipo de imágenes son especialmente útiles para estudiar pequeños municipios aislados, urbanizaciones, etc.

Figura 9.9: Benalmádena'99.

Figura 9.10: Benalmádena'00.

9.3 Pruebas

Como hemos comentado en la sección 9.1, las baterías de pruebas se han diseñado atendiendo fundamentalmente a las etapas del proceso de detección de cambios:

1. Corrección geométrica
2. Corrección radiométrica
3. Diferencia de imágenes

9.3.1 Corrección geométrica

Las pruebas diseñadas para la etapa de corrección geométrica tienen como objetivo demostrar la eficacia de la técnica empleada para implementarla y su comportamiento ante imágenes con diferentes contenidos. Trataremos básicamente de probar los siguientes aspectos:

- Búsqueda de GCPs
- Seguimiento de GCPs
- Política de eliminación de GCPs
- Polinomio de ajuste
- Corrección local mediante el uso POIs (polígonos de interés)

9.3.2 Corrección radiométrica

Las pruebas diseñadas para la etapa de corrección radiométrica tienen como objetivo demostrar la eficacia de la técnica empleada para implementarla. Trataremos básicamente de probar el siguiente aspecto:

- Especificación del histograma

9.3.3 Diferencia de imágenes

La técnica empleada en el proceso de detección de cambios es tan sencilla como eficaz, por tanto, diseñamos pruebas que nos permiten estudiar el histograma de las imágenes de cambios procesadas.

- Valor absoluto
- Desplazamiento

Del estudio de estos histogramas se puede extraer información relativa a los umbrales utilizados en el proceso de segmentación de la imagen, con objeto de extraer un mapa de cambios.

Capítulo 10

Resultados

10.1 Introducción

En este capítulo se presentan los resultados de las pruebas descritas en el capítulo anterior, mostraremos las imágenes resultado obtenidas y los diferentes datos relacionados con las distintas etapas del proceso de detección de cambios.

Naturalmente, y continuando con la tendencia seguida hasta el momento, dividiremos los resultados de las pruebas atendiendo a las etapas del proceso de detección. Comenzando con los resultados de las pruebas realizadas para validar la eficacia de la técnica de corrección geométrica y finalizando con el estudio de los histogramas que presentan las imágenes de cambios procesadas, junto con las imágenes resultado del proceso de umbralización.

10.2 Corrección geométrica

En la sección 9.3 se detallaban los diferentes aspectos a considerar a la hora de determinar la bondad de la técnica empleada para corregir geométricamente pares de imágenes.

- Búsqueda de GCPs
- Seguimiento de GCPs
- Política de eliminación de GCPs

- Polinomio de ajuste
- Corrección local mediante el uso POIs (polígonos de interés)

Los campos modificados de la estructura `KLT_TrackingContext` son los siguientes:

1. `int window_width = 15;`
2. `int window_height = 15;`
3. Ejecutar `KLTUpdateTCBorder(KLT_TrackingContext tc)` para actualizar los campos `int borderx, bordery;`

El número máximo de características exigidas es de 25. Reducir este último valor puede provocar un elevado número de tiles sin característica, es primordial, siempre que sea posible, asegurar una característica por tile.

10.2.1 Global

Hablamos de corrección geométrica *global* cuando el polígono de interés (POI) abarca la totalidad de las imágenes.

Información

Las imágenes utilizadas son las pertenecientes a la serie multi-temporal de Málaga. A continuación presentamos información relacionada con las imágenes utilizadas.

1. **Dimensión de la imagen:** 512×512 píxeles
2. **Niveles de gris:** 256 (0-255)
3. **Dimensión del tile:** 128×128 píxeles
4. **Número de tiles verticales:** 4
5. **Número de tiles horizontales:** 4
6. **Número de tiles:** 16 (4×4)

7. **% de solape:** 94%

8. **Polígono de interés:** Ninguno

El ajuste del motor de búsqueda y seguimiento de características se ha ajustado para el tamaño de tile especificado.

Búsqueda de GCPs

En la figura 10.1 podemos observar el resultado de la búsqueda de GCPs en la imagen de referencia (Málaga'99). Los GCPs localizados se identifican mediante cruces (+).

Figura 10.1: Búsqueda de GCPs en Málaga'99.

La tabla 10.1 contiene las coordenadas de los GCPs localizados en la

figura 10.1, aquellos tiles para los que no han sido localizados ningún GCP presentan un -1 en las columnas correspondientes a las coordenadas.

Tile		**GCP**	
x	**y**	**x**	**y**
0	0	80.000000	51.000000
0	1	193.000000	62.000000
0	2	292.000000	55.000000
0	3	485.000000	45.000000
1	0	-1.000000	-1.000000
1	1	221.000000	159.000000
1	2	357.000000	165.000000
1	3	477.000000	157.000000
2	0	53.000000	286.000000
2	1	-1.000000	-1.000000
2	2	-1.000000	-1.000000
2	3	444.000000	286.000000
3	0	103.000000	408.000000
3	1	190.000000	483.000000
3	2	349.000000	426.000000
3	3	-1.000000	-1.000000

Tabla 10.1: GCPs detectados en la figura 10.1.

Seguimiento de GCPs

En la figura 10.1 podemos observar el resultado del seguimiento de GCPs en la imagen a corregir (Málaga'00). Los GCPs seguidos se identifican mediante cruces (+).

Figura 10.2: Seguimiento de GCPs en Málaga'99.

La tabla 10.2 contiene las coordenadas de los GCPs seguidos en la figura 10.2 y los residuos, aquellos tiles para los que no han sido seguidos ningún GCP presentan un -1 en las columnas correspondientes a las coordenadas.

Tile		GCP		Residuos	
x	y	x	y	cx	cy
0	0	64.911484	60.801441	15.088516	-9.801441
0	1	181.777561	46.557648	11.222439	15.442352
0	2	288.436588	49.631901	3.563412	5.368099
0	3	469.643219	52.944145	15.356781	-7.944145
1	0	-1.000000	-1.000000	0.000000	0.000000
1	1	205.408981	170.568642	15.591019	-11.568642
1	2	341.301376	173.089973	15.698624	-8.089973
1	3	461.327713	165.482704	15.672287	-8.482704
2	0	38.073761	297.015652	14.926239	-11.015652
2	1	-1.000000	-1.000000	0.000000	0.000000
2	2	-1.000000	-1.000000	0.000000	0.000000
2	3	428.203423	294.332935	15.796577	-8.332935
3	0	88.773430	416.773643	14.226570	-8.773643
3	1	193.149437	473.766220	-3.149437	9.233780
3	2	335.676857	434.625546	13.323143	-8.625546
3	3	-1.000000	-1.000000	0.000000	0.000000

Tabla 10.2: GCPs seguidos en la figura 10.2.

El número teórico de GCPs es 16, una vez procesada la imagen, el número de GCPs localizados y seguidos es 12, hablamos, por tanto, de un 75% de efectividad.

Política de eliminación de GCPs

Como comentamos en la sección 6.2, una vez localizados y seguidos los pares de GCPs en las imágenes de la serie multi-temporal, el siguiente paso consiste en eliminar los GCPs que provocan que el error cuadrático medio (RMS) sea superior a un píxel (política de eliminación de GCPs). El mejor candidato para su eliminación, es aquel GCP que presenta el mayor error longitudinal (EL).

La tabla 10.3 contiene los GCPs eliminados, junto con sus respectivos residuos y errores longitudinales. Los GCPs eliminados se identifican mediante asteriscos (∗), ver figuras 10.1 y 10.2.

Tile		GCP		Residuos		
x	**y**	**x**	**y**	**cx**	**cy**	**EL**
3	1	193.149437	473.766220	-3.149437	9.233780	19.947901
0	1	181.777561	46.557648	11.222439	15.442352	15.969753
0	2	288.436588	49.631901	3.563412	5.368099	14.465020
1	1	205.408981	170.568642	15.591019	-11.568642	1.937016

Tabla 10.3: GCPs eliminados en las figuras 10.1 y 10.2.

El número de GCPs localizados y seguidos en las imágenes de la serie multi-temporal se reduce a 8, el porcentaje desciende hasta el 50%. Un porcentaje suficiente para atacar con garantías de éxito la corrección geométrica de imágenes de 512×512 píxeles.

> **NOTA:** Podemos observar como los GCPs eliminados se encuentran ordenados de mayor a menor EL.

Polinomio de ajuste

Una vez eliminados aquellos GCPs que provocan un desajuste superior al píxel, la siguiente etapa de la corrección geométrica consiste en la transferencia de niveles de gris de la imagen original a la imagen corregida. Los encargados de definir las nuevas coordenadas de cada uno de los píxeles de la imagen corregida son los polinomios de ajuste. En la siguiente serie de imágenes comprobaremos la potencia de la técnica empleada, capaz de corregir imágenes con deformaciones afines muy superiores a las que habitualmente se producen en la adquisición de las imágenes de satélite.

Traslación (10 píxeles ↓ y 10 píxeles →)

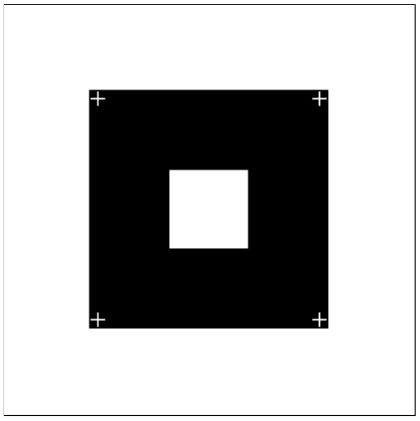


Figura 10.3: Original.

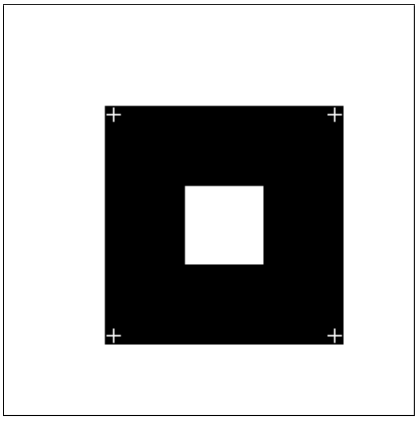


Figura 10.4: Traslación.

Polinomios de ajuste

$$\begin{aligned}\hat{x} = \; & -9.997825 + 1.000319x + \\ & +0.000254y - 0.000001xy \\ \hat{y} = \; & -9.997825 + 0.000254x + \\ & +1.000319y - 0.000001xy\end{aligned}$$

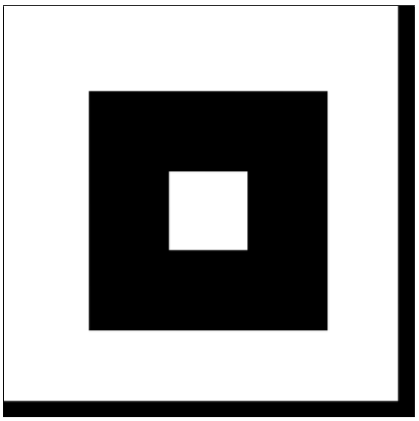


Figura 10.5: Traslación corregida.

Escalado (10 píxeles →← y 10 píxeles ↓↑)

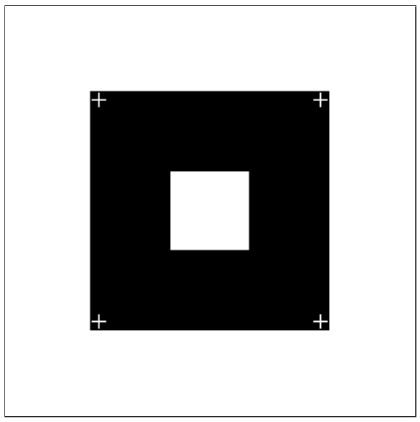


Figura 10.6: Original.

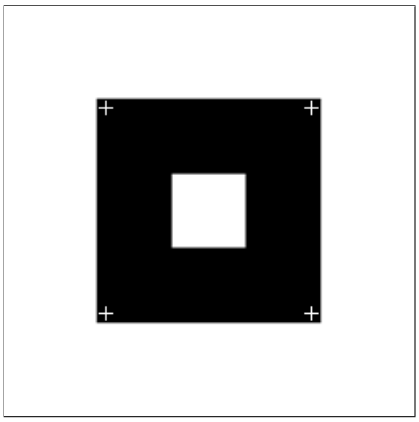


Figura 10.7: Escalado.

Polinomios de ajuste

$$\begin{aligned}\hat{x} = \; & -9.221236 + 1.072329x + \\ & -0.000009y + 0.000000xy \\ \hat{y} = \; & -9.222414 - 0.000006x + \\ & +1.072340y + 0.000000xy\end{aligned}$$

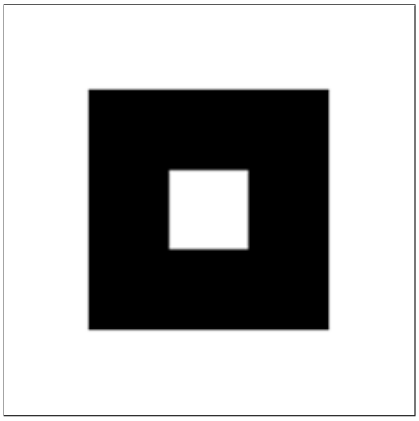


Figura 10.8: Escalado corregida.

Inclinación (15 píxeles $\rightarrow$ y 15 píxeles $\leftarrow$)

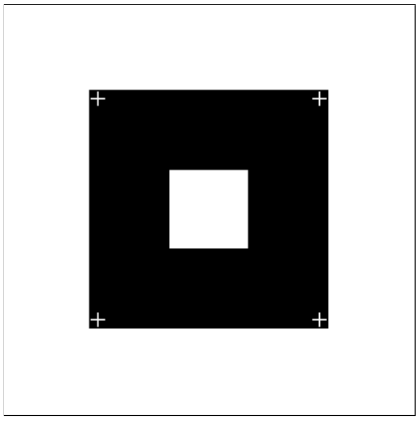


Figura 10.9: Original.

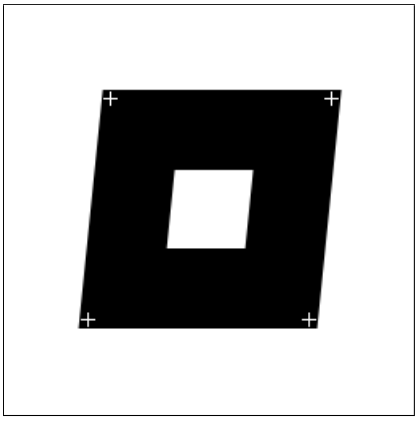


Figura 10.10: Inclinación.

Polinomios de ajuste

$$\begin{aligned}\hat{x} = \; & -13.640133 + 1.000198x + \\ & +0.098555y + 0.000000xy \\ \hat{y} = \; & -0.159261 + 0.000988x + \\ & +1.000266y - 0.000000xy\end{aligned}$$

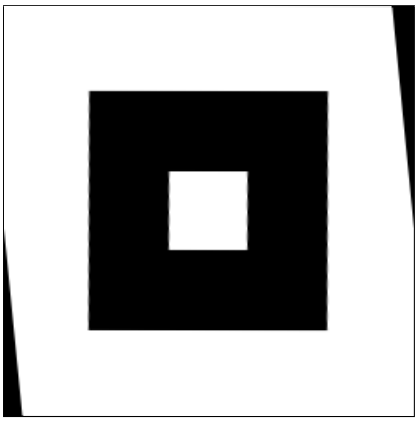


Figura 10.11: Inclinación corregida.

Perspectiva (10 píxeles $^{\rightarrow\leftarrow}$ y 10 píxeles $_{\leftarrow\rightarrow}$)

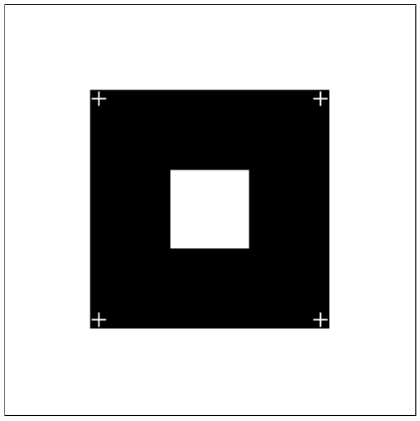


Figura 10.12: Original.

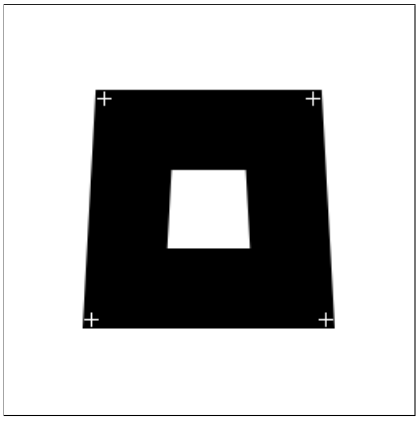


Figura 10.13: Perspectiva.

Polinomios de ajuste

$$\begin{aligned}\hat{x} = \; & -12.784172 + 1.100662x + \\ & +0.097030y - 0.000764xy \\ \hat{y} = \; & -0.049502 - 0.000014x + \\ & +1.000118y - 0.000000xy\end{aligned}$$

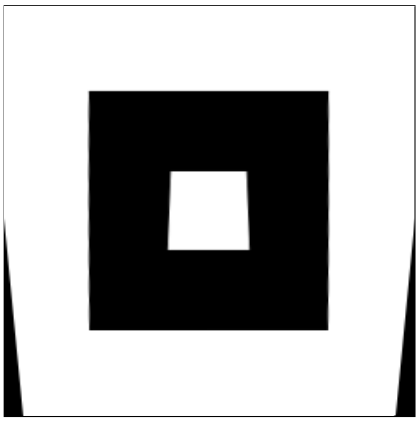


Figura 10.14: Perspectiva corregida.

Rotación (5 grados ↷)

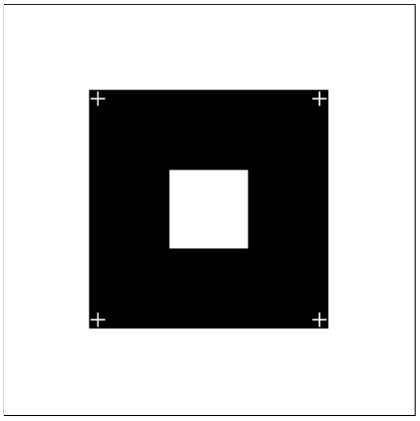


Figura 10.15: Original.

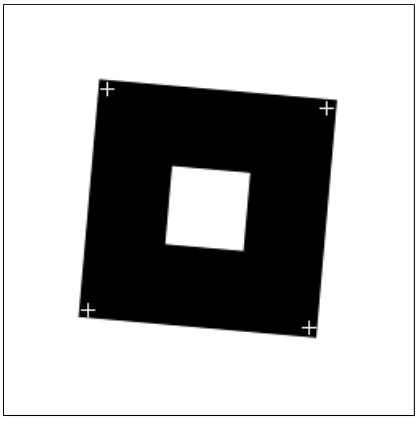


Figura 10.16: Rotación.

Polinomios de ajuste

$$\begin{aligned}\hat{x} = \; & -9.896312 + 0.996519x + \\ & +0.083709y + 0.000001xy \\ \hat{y} = \; & 11.445890 - 0.083709x + \\ & +0.996699y - 0.000000xy\end{aligned}$$

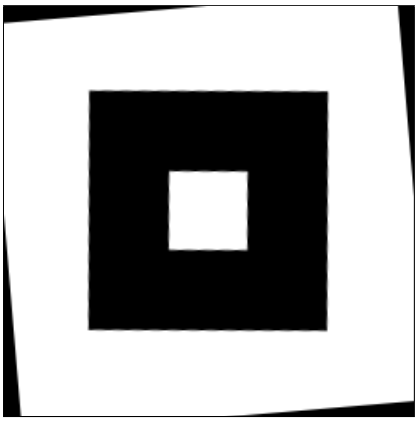


Figura 10.17: Rotación corregida.

Por último, los polinomios de ajuste calculados a partir del conjunto de GCPs localizados y seguidos en las figuras 10.1 y 10.2 quedarían de la siguiente forma:

$$\begin{aligned} \hat{x} = &\quad 15.432620 + 1.001621x - 0.002956y - 0.000003xy \\ \hat{y} = &\quad -10.509747 + 0.005384x + 1.001434y - 0.000004xy \end{aligned}$$

La figura 10.18 muestra la imagen corregida geométricamente, en ella se puede observar un claro desplazamiento en diagonal (↗)

Figura 10.18: Málaga'00 corregida geométricamente.

Método de transferencia de niveles de gris

Los métodos a utilizar en la transferencia de los niveles de gris de la imagen original a la imagen corregida son tres: a) vecino más próximo, b) interpolación bilinear y c) convolución cúbica. La elección de uno u otro método dependerá fundamentalmente del tipo de aplicación. En las figuras 10.19 y 10.20 se puede observar gráficamente el porqué de la decisión tomada en la sección 6.2 sobre el método de trasferencia a utilizar.

Figura 10.19: Vecino más próximo.

Figura 10.20: Convolución cúbica.

Como comentamos en su momento, la ventaja del método del *vecino más próximo* era que no modificaba los niveles de gris de la imagen corregida, pero el efecto que produce (fragmentación de los bordes, líneas, etc.) es nefasto para el tipo de aplicación que nos ocupa. Es preferible que los niveles de gris de la imagen corregida se vean ligeramente afectados si el método de trasvase no produce deformaciones tales, que puedan ser identificadas en etapas posteriores como cambios, por eso optamos por el método de la *convolución cúbica*.

10.2.2 Local

Hablamos de corrección geométrica *local* cuando el polígono de interés (POI) restringe el área de corrección a una región inferior a la totalidad de la imagen.

Información

Las imágenes utilizadas son las pertenecientes a la serie multi-temporal de Benalmádena. A continuación presentamos información relacionada con las imágenes utilizadas.

1. **Dimensión de la imagen:** 512×512 píxeles
2. **Niveles de gris:** 256 (0-255)
3. **Dimensión del tile:** 128×128 píxeles
4. **Número de tiles verticales:** 4
5. **Número de tiles horizontales:** 4
6. **Número de tiles:** 16 (4×4)
7. **% de solape:** 99%
8. **Polígono de interés:**

Vértice	
x	**y**
86	66
241	63
463	314
442	446
308	443
51	186
86	66

Tabla 10.4: Polígono de interés.

> **NOTA:** Si dibujáramos el polígono de la tabla 10.4 sobre la imagen, el polígono abarcaría la región que coincide con la diagonal principal de la imagen. Otro aspecto a considerar a la hora de especificar el polígono de interés es que debe ser cerrado (los vértices inicial y final son iguales).

El ajuste del motor de búsqueda y seguimiento de características se ha ajustado para el tamaño de tile especificado.

Búsqueda de GCPs

En la figura 10.21 podemos observar el resultado de la búsqueda de GCPs en la imagen de referencia (Benalmádena'99). Los GCPs localizados se identifican mediante cruces (+).

Figura 10.21: Búsqueda de GCPs en Benalmádena'99.

La tabla 10.5 contiene las coordenadas de los GCPs localizados en la figura 10.21, aquellos tiles para los que no han sido localizados ningún GCP presentan un -1 en las columnas correspondientes a las coordenadas, aquellos otros que están fuera del polígono de interés descrito en la tabla 10.4 presentan un -2.

Tile		GCP	
x	y	x	y
0	0	58.000000	24.000000
0	1	213.000000	24.000000
0	2	286.000000	97.000000
0	3	-2.000000	-2.000000
1	0	27.000000	162.000000
1	1	191.000000	168.000000
1	2	-1.000000	-1.000000
1	3	462.000000	152.000000
2	0	102.000000	337.000000
2	1	214.000000	357.000000
2	2	336.000000	311.000000
2	3	466.000000	318.000000
3	0	-2.000000	-2.000000
3	1	177.000000	410.000000
3	2	-1.000000	-1.000000
3	3	439.000000	420.000000

Tabla 10.5: GCPs detectados en la figura 10.21.

Seguimiento de GCPs

En la figura 10.21 podemos observar el resultado del seguimiento de GCPs en la imagen a corregir (Benalmádena'00). Los GCPs seguidos se identifican mediante cruces (+).

Figura 10.22: Seguimiento de GCPs en Benamádena'99.

La tabla 10.6 contiene las coordenadas de los GCPs seguidos en la figura 10.2 y los residuos, aquellos tiles para los que no han sido seguidos ningún GCP presentan un -1 en las columnas correspondientes a las coordenadas, aquellos otros que están fuera del polígono de interés descrito en la tabla 10.4 presentan un -2.

Tile		GCP		Residuos	
x	**y**	**x**	**y**	**cx**	**cy**
0	0	59.214520	25.335791	-1.214520	-1.335791
0	1	213.974052	24.493132	-0.974052	-0.493132
0	2	287.878616	97.152946	-1.878616	-0.152946
0	3	-2.000000	-2.000000	0.000000	0.000000
1	0	26.747725	162.315189	0.252275	-0.315189
1	1	191.158039	168.616608	-0.158039	-0.616608
1	2	-1.000000	-1.000000	0.000000	0.000000
1	3	462.728760	152.873667	-0.728760	-0.873667
2	0	102.204094	337.186264	-0.204094	-0.186264
2	1	214.302597	358.007332	-0.302597	-1.007332
2	2	337.425804	312.096420	-1.425804	-1.096420
2	3	467.253517	318.139874	-1.253517	-0.139874
3	0	-2.000000	-2.000000	0.000000	0.000000
3	1	177.157261	410.812302	-0.157261	-0.812302
3	2	-1.000000	-1.000000	0.000000	0.000000
3	3	439.493221	420.140324	-0.493221	-0.140324

Tabla 10.6: GCPs detectados en la figura 10.22.

El número teórico de GCPs es 14 (excluidos los tiles situados fuera del polígono de interés), una vez procesada la imagen, el número de GCPs localizados y seguidos es 12, hablamos, por tanto, de un 85% de efectividad.

Política de eliminación de GCPs

En este caso y a diferencia del ejemplo anterior (serie multi-temporal de Málaga), el número de GCPs eliminados es cero, hablamos por tanto de un 100% de efectividad sobre el número de GCPs localizados y seguidos.

El hecho de utilizar polígonos para localizar la región que deseamos corregir nos permite calcular eficientemente los polinomios de ajuste, de forma que no sean considerados en el proceso de búsqueda y seguimiento, aquellos tiles que muy probablemente podrían introducir GCPs de escaso interés (mar, montañas, etc.) para nuestro propósito, acortando el proceso de corrección y evitando pequeñas desviaciones en el cálculo.

Polinomio de ajuste

La siguiente etapa de la corrección geométrica consiste en la transferencia de niveles de gris de la imagen original a la imagen corregida. Los encargados de definir las nuevas coordenadas de cada uno de los píxeles de la imagen corregida son los polinomios de ajuste.

$$\begin{aligned} \hat{x} = &\quad -0.810088 + 0.998345x + 0.003114y - 0.000003xy \\ \hat{y} = &\quad -0.797572 + 0.000483x + 1.000364y - 0.000000xy \end{aligned}$$

La figura 10.23 muestra la imagen corregida geométricamente, en ella se puede observar una ligera rotación ($\curvearrowleft$).

Figura 10.23: Benalmádena'00 corregida geométricamente.

10.3 Corrección radiométrica

En la sección 9.3 se detallaban los diferentes aspectos a considerar a la hora de determinar la bondad de la técnica empleada para corregir radiométricamente pares de imágenes.

- Especificación del histograma

Como comentamos en la sección 6.3 son muchos los procedimientos utilizados para corregir radiométricamente una imagen, o más concretamente, normalizar el nivel de gris la imagen a corregir con respecto a la de referencia. Sin embargo, tan sólo uno de ellos facilitaba la automatización, este procedimiento era la *especificación del histograma*.

Información

Las imágenes utilizadas son las pertenecientes a la serie multi-temporal de Benalmádena. A continuación presentamos información relacionada con las imágenes utilizadas.

1. **Dimensión de la imagen:** 512×512 píxeles
2. **Niveles de gris:** 256 (0-255)
3. **% de solape:** 99%

Especificación del histograma

El proceso de normalización radiométrica utilizando la especificación del histograma consiste en calcular el histograma de la imagen de referencia y utilizar dicho histograma para especificar el histograma de la imagen a corregir. Para una desarrollo matemático de la especificación del histograma ver la sección 6.3.

En las figuras 10.25, 10.27 y 10.29 podemos observar gráficamente el proceso de especificación y los resultados obtenidos de dicho procesamiento.

Histograma de la imagen de referencia

Figura 10.24: Benalmádena'99.

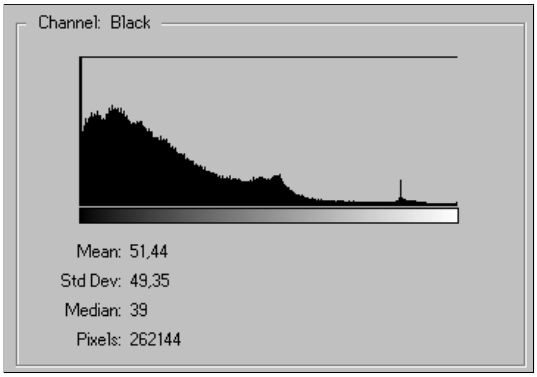


Figura 10.25: Histograma de Benalmádena'99.

Histograma de la imagen a corregir

Figura 10.26: Benalmádena'00.

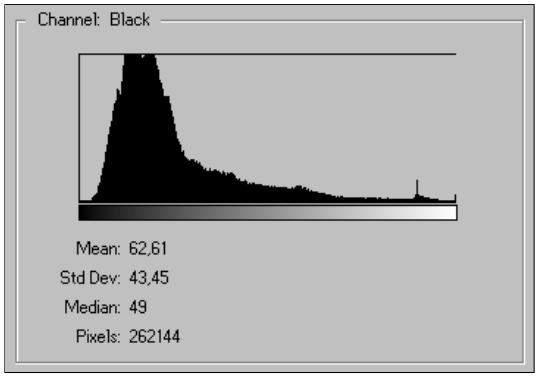


Figura 10.27: Histograma de Benalmádena'00.

Histograma de la imagen corregida

Figura 10.28: Benalmádena'00 corregida radiométricamente.

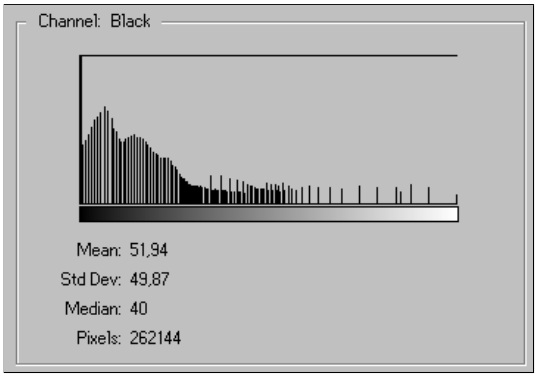


Figura 10.29: Histograma de Benalmádena'00 corregida radiométricamente.

El aspecto "quebrado" del histograma de la imagen corregida se debe al tamaño relativamente pequeño de la imagen a corregir, en imágenes con dimensiones superiores a 1024×1024 el aspecto del histograma es sorprendentemente parecido al de la imagen de referencia.

10.4 Diferencia de imágenes

Como ya comentamos en la sección 9.3, la técnica empleada en el proceso de detección de cambios es tan sencilla como eficaz, por tanto, mostraremos el resultado de dicho procesamiento y estudiaremos el histograma de las imágenes obtenidas para realizar con garantías de éxito el proceso de umbralización.

Ya expusimos en la sección 6.4 las ventajas de los dos tipos de procesamiento: a) valor absoluto y b) desplazamiento, y su especial utilidad a la hora de determinar los umbrales para obtener los mapas de cambios.

En las figuras 10.30, 10.33, 10.31 y 10.34 podemos observar las imágenes procesadas junto con sus respectivos histogramas.

Histograma de la imagen diferencia - Valor absoluto

Figura 10.30: Imagen diferencia - Valor absoluto.

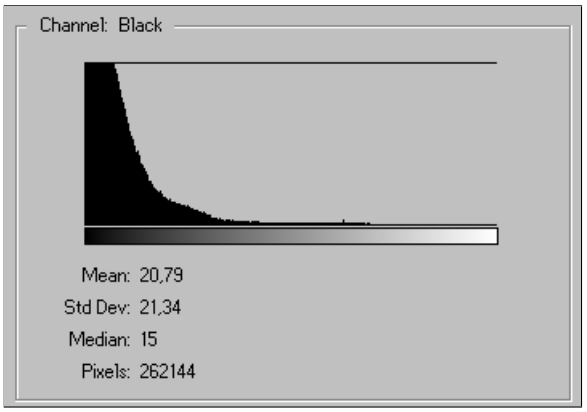


Figura 10.31: Histograma de la imagen diferencia - Valor absoluto.

En la figura 10.32 podemos observar la imagen diferencia una vez procesada y umbralizada con un umbral t=80. La elección del umbral es crucial a la hora de determinar el nivel de cambio que deseamos estimar.

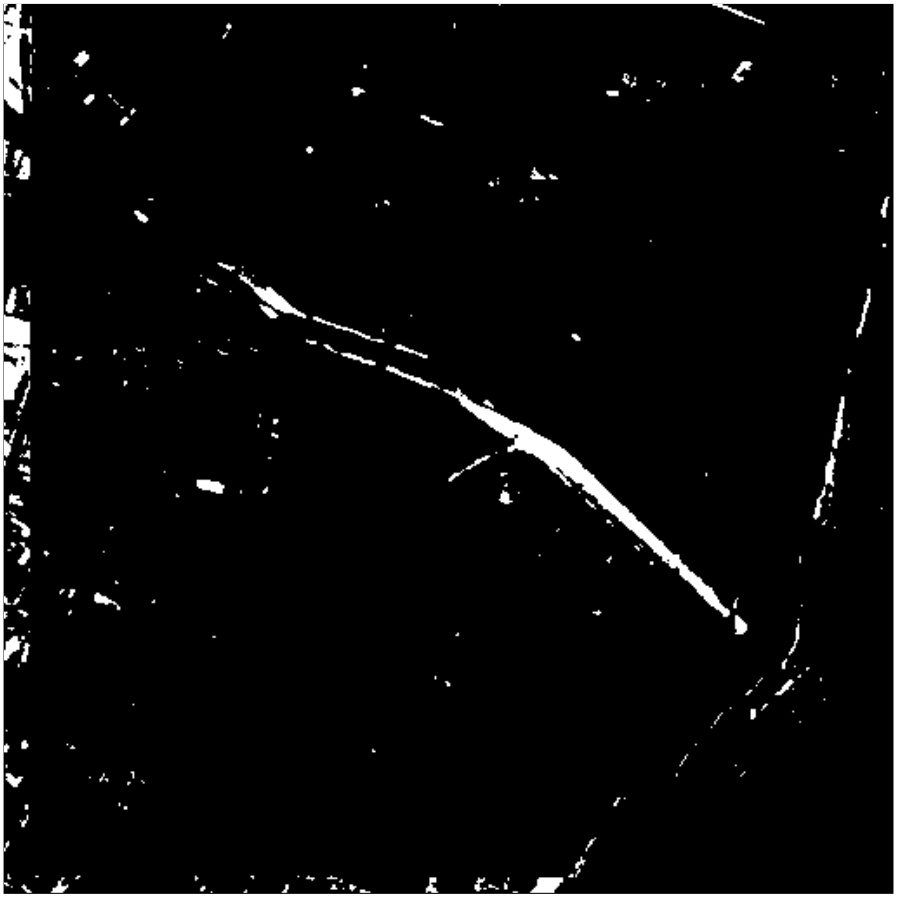


Figura 10.32: Imagen diferencia - Cambios (t=80).

La mayoría de las aplicaciones de detección de cambios emplean este tipo de procesamiento, puesto que te permite determinar la existencia del cambio con independencia de su sentido. Si además de determinar la existencia de un cambio, necesitamos aportar información sobre el sentido del mismo es imprescindible trabajar con desplazamientos (offsets). Los desplazamientos, como su propio nombre indica, desplazan el histograma y lo centran en el valor especificado.

Histograma de la imagen diferencia - Desplazamiento (o=127)

Figura 10.33: Imagen diferencia - Desplazamiento (o=127).

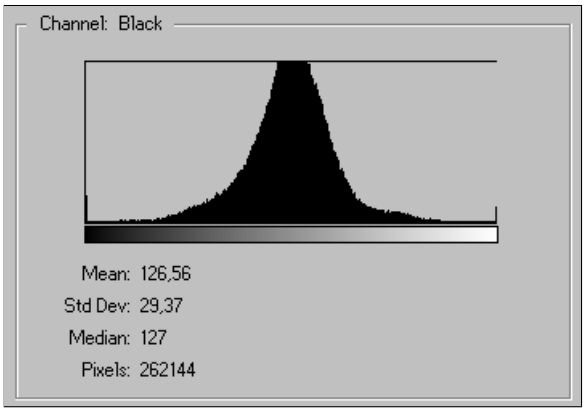


Figura 10.34: Histograma de la imagen diferencia - Desplazamiento (o=127).

En la figura 10.35 podemos observar la imagen diferencia una vez procesada y umbralizada con un umbral t=200. Como resultado obtenemos una imagen en la que se aprecian aquellos cambios que han producido en la imagen a corregir y que no existían en la imagen de referencia (cambios positivos)

Figura 10.35: Imagen diferencia - Cambios positivos (t=200).

En la figura 10.36 podemos observar la imagen diferencia una vez procesada y umbralizada con un umbral t=65. Como resultado obtenemos una imagen en la que se aprecian aquellos cambios que existían en la imagen de referencia y que no se han producido en la imagen a corregir (cambios negativos).

> **NOTA:** La imagen se encuentra invertida, es decir los cambios aparecen en negro sobre fondo blanco y no en blanco sobre fondo blanco como hasta ahora.

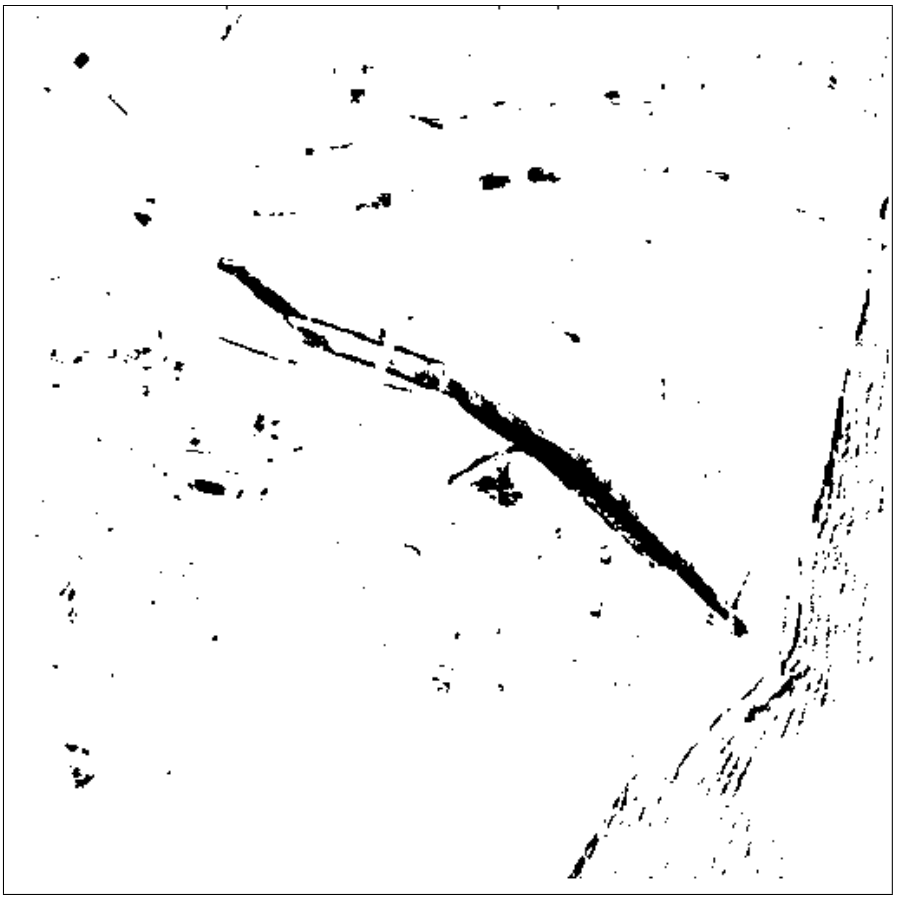


Figura 10.36: Imagen diferencia - Cambios negativos (t=65).

Parte IV

Conclusiones y futuras mejoras

Capítulo 11

Conclusiones

11.1 Introducción

En este capítulo detallaremos los pros y contras de nuestro proyecto, los problemas con los que nos hemos encontrado y como no, si hemos o no alcanzado los objetivos que se establecieron en un principio.

11.2 Pros y contras

Estoy especialmente contento porque el número de pros supera sensiblemente el de contras, aunque sencillamente, todo se puede mejorar.

11.2.1 Pros

El resultado final del proyecto se ajusta a las especificaciones que se establecieron en un principio, también se ajusta a lo que demandó en su día la gerencia de urbanismo del Iltmo. Ayuntamiento de Málaga, aporta soluciones nuevas que permiten automatizar etapas que hasta ahora se venían realizando con un elevado grado de intervención humana. Este último aspecto es, personalmente, es el mayor logro de este proyecto. Un conjunto de aplicaciones que permiten realizar un proceso que en la mayor parte de los sistemas estudiados requiere de gran cantidad de intervención humana.

Con respecto a las técnicas empleadas en el desarrollo de cada uno de las etapas del proceso de detección de cambios, he de destacar la destinada

a corregir geométricamente las imágenes. Esta técnica ha sido desarrollada íntegramente por el autor, en colaboración con el Dr. Javier González Jiménez, de hecho, los resultados han sido tan satisfactorios que han sido presentados en el congreso Urban'2001[1][9] que se celebrará en Roma el próximo 8 de noviembre. Como ya hemos comentado en capítulos anteriores, donde se presentaba la técnica y los recursos software empleados, el éxito de esta técnica radica fundamentalmente en la adaptación de un algoritmo que fue inicialmente diseñado para el seguimiento de características en aplicaciones de visión estéreo (KLT feature tracker) para localizar y seguir GCPs en imágenes de satélite.

Tampoco podemos olvidar la técnica empleada en la etapa de corrección radiométrica, con ella también innovamos, en base al estudio previo realizado antes de comenzar el proyecto, ninguno de los proyectos a los que hemos tenido acceso, utilizan esta técnica para normalizar radiométricamente las imágenes de la serie multi-temporal, como norma general suelen utilizar ajustes absolutos (conversión a reflectividades, radiancias, etc.), nosotros hemos ido un poco más allá, proporcionando un enfoque relativo a la etapa de corrección radiométrica[2].

La técnica empleada en el proceso de detección de cambios, quizás no sea tan elaborada como algunas de las presentadas, pero si las cosas funcionan ¿para qué cambiar?. Es la más utilizada en la mayoría de los aplicaciones de detección de cambios, es sencilla y eficaz. Los procesamientos adicionales permiten explotar y mejorar los resultados proporcionados por esta técnica.

También se ha desarrollado una librería de clases en ANSI C++ que constituye un gran punto de partida para futuras aplicaciones. Optimiza el uso de recursos disponibles en el sistema, simultaneando gestión de disco y memoria. Implementa algoritmos fácilmente extrapolables a problemas similares (cálculo de coeficientes, política de eliminación de puntos no válidos, etc.).

Por último, cabe destacar la estructura de los módulos desarrollados, de

[1] IEEE/ISPRS joint Workshop on Remote Sensing and Data Fusion over Urban Areas

[2] Para que utilizar parámetros proporcionados por el satélite sujetos en la mayoría de los casos a error, pudiendo ajustar los niveles de gris de la imagen a corregir con respecto a la de referencia.

tal forma que pueden ser fácilmente integrados en aplicaciones disponibles actualmente en el mercado (GRASS). Este aspecto, a simple vista, sin importancia, agiliza el tratamiento de los datos suministrados por nuestros módulos de tal forma que pueden formar parte de un sistema información geográfica (GIS) sin un coste de adaptación demasiado elevado.

11.2.2 Contras

Nuestro sistema no es perfecto, y como tal, hay ciertos aspectos que se podrían haber mejorado. Técnicamente, las técnicas empleadas son las adecuadas y los resultados así lo demuestran, pero desde el punto de vista de la implementación hubiera sido interesante que la aplicación no dependiese de la máquina sobre la que se utiliza. Como ya comentamos en su momento, estas aplicaciones encapsulan ciertas funciones íntimamente relacionadas con el tipo de microprocesador que posee la máquina sobre la que corre, hemos utilizado funciones de la Intel Image Processing Library 2.5 especialmente optimizadas para sus propios microprocesadores. Si deseamos extraer el máximo rendimiento de los módulos desarrollados, deben haber sido desarrollados con la librerías adecuadas. Esto limita, en cierta medida, las posibilidades de ejecución en otras máquinas y dificulta su portado a otros sistemas (Solaris, etc.).

El sistema no incorpora técnicas para trabajar con mosaicos de imágenes, las imágenes con las que hemos venido trabajando tienen una resolución espacial de 5 metros, en un futuro cercano (muy cercano) estarán disponibles comercialmente (a un precio asequible[3]) imágenes con resoluciones espaciales de 1 metro e incluso inferiores, estamos hablando de imágenes con coberturas mucho menores y que para cubrir el área de estudio nos obligaría a trabajar con grupos de ellas.

11.3 Problemas

Uno de los problemas con los que nos hemos encontrado a lo largo de todo el proyecto ha sido la falta de información. Las tareas de documentación

[3]Actualmente, existen pero a un precio muy elevado

han sido arduas y laboriosas, la detección de cambios urbanos a partir de imágenes de satélite es un campo aplicación incipiente, y no son muchas las fuentes de información en las que se detallan procedimientos o técnicas para afrontar las distintas etapas del proceso.

Capítulo 12

Futuras mejoras

12.1 Introducción

En este capítulo detallaremos las futuras mejoras que se pueden introducir en futuros proyectos. Para ello atenderemos a las técnicas de procesamiento, a la implementación, etc.

12.2 Técnicas

Como comentamos en el el capítulo anterior las técnicas empleadas han proporcionado unos resultados excelentes, sin embargo es necesario considerar aspectos que muy probablemente sean necesarios en un futuro no muy lejano. Entre ellos cabría destacar la utilización de algoritmos de gestión de *mosaicos* de imágenes, con objeto de gestionar *clusters* de imágenes con una mayor resolución espacial pero con un área de cobertura inferior a la actual.

Con respecto a las técnicas empleadas en cada una de las etapas del proceso de detección, sería conveniente desarrollar algún tipo de procesamiento alternativo capaz de recuperar GCPs en aquellos tiles que por algún motivo carecen de él, bien por que no haya sido detectado en el proceso de búsqueda o por que simplemente ha sido eliminado en el proceso de ajuste.

Con objeto de mejorar la normalización radiométrica de las imágenes que intervienen en el proceso de detección sería conveniente realizar una especificación local del histograma en lugar de global. Este aspecto es importante, ya que las condiciones de iluminación son variables en toda la imagen. Es-

to, unido a la corrección geométrica local, nos plantea otro posible enfoque de todo el proceso de detección. Abordar el proceso de detección de forma local, sobre regiones de interés y no sobre la totalidad de la imagen. Ya hemos dado un paso en este sentido con la técnica de corrección geométrica empleada (polígonos de interés), pero sería interesante extender el concepto de localidad a la corrección radiométrica.

Sería interesante dotar al sistema de un sistema de clasificación capaz de determinar si los cambios producidos se pueden clasificar como cambios urbanos o simples accidentes del terreno. Es decir, dotar al sistema de la capacidad de discriminar entre positivos (construcción ilegal) y falsos positivos (movimientos de tierra) utilizando para ello criterios tales como: área, forma, etc. del cambio, esto proporcionaría un enorme valor añadido a la aplicación.

12.3 Implementación

Resulta interesante la idea de incorporar cada uno de los módulos a aplicaciones comerciales ya contrastadas, sin embargo, estas aplicaciones te marcan unas pautas y en cierto modo, aunque puedes aprovechar las herramientas que incorporan, restringen las posibilidades de la aplicación. Personalmente, una interesante mejora del sistema consistiría en desarrollar su propio *front end* que facilitase la creación intuitiva de áreas de interés, la gestión de múltiples fuentes de datos (recursos cartográficos, etc.), etc. y permitiese su uso coordinado para proporcionar una solución global.

Como comentamos en el capítulo anterior, uno de los inconvenientes de los módulos desarrollados, es su ligazón con la máquina sobre la que se utiliza. Otra futura mejora consistiría en desligar la librería de clases de la máquina sobre la que va correr utilizando primitivas 100% estándar, de forma que se mejore su portabilidad a otros sistemas.

Parte V

Referencias y bibliografía

Referencias

[1] P. E. Anuta. "Spatial registration of multispectral and multitemporal digital imagery using FFT techniques". *IEEE Transactions on Geoscience and Electronics*, 8(1):353–368, 1970.

[2] D. I. Barnea and H. F. Silverman. "A class of algorithms for fast digital image registration". *IEEE Transactions on Computers*, 21(1):179–186, 1972.

[3] J. S. Boland and H. S. Ranganath. *Automatic handoff of multiple targets.* Tech. Rep. Auburn Univ., primera edition, 1980.

[4] Emilio Chuvieco. *Fundamentos de teledetección espacial.* Ediciones RIALP, tercera edition, 2000.

[5] J. R. Eastman, J. McKendry, and M. Fulk. *UNITAR Explorations in GIS technology: Change and time series analysis*, volume 1. Geneve, Switzerland, primera edition, 1994.

[6] C. D. Elvidge, D. Yuan, R. D. Weerackoon, and R. S. Lunneta. "Relative radiometric normalization of Landsat Multispectral Scanner (MSS) data using an automatic scattergram-controlled regression". *ISPRS J. Photogramm. Remote Sensing*, 61(10):1255–1260, 1995.

[7] Jan Flusser and Tomáš Suk. "A moment-based approach to registration of images with affine geometric distortion". *IEEE Transactions on Geoscience and Remote Sensing*, 32(2):382–387, 1994.

[8] Canada Centre for Remote Sensing. CCRS Remote sensing tutorial. `http://www.ccrs.nrcan.gc.ca/ccrs/eduref/tutorial/tutore.html`, Enero 2001.

[9] Javier González-Jiménez, Gregorio Ambrosio, and Vicente Arévalo. Automatic urban change detection from the irs-1d pan. In *(Enviado a) IEEE-ISPRS Joint Workshop on Remote Sensing and Data Fusion over Urban Areas*, pages 320–323, Rome, Italy, November 2001.

[10] Javier González Jiménez. *Visión por computador*. ITP-Paraninfo, primera edition, 1999.

[11] Object Management Group. *OMG Unified Modeling Language Specification, ver. 1.3*. Object Management Group, 1999.

[12] Diem Ho and Adel Asem. "NOAA AVHRR image referencing". *Int. J. Remote Sensing*, 7(7):895–904, 1986.

[13] Bruce D. Lucas and Takeo Kanade. "An iterative image registration technique with an application to stereo vision". *International Joint Conference on Artificial Intelligence*, pages 674–679, 1981.

[14] D. A. Mouat, G. G. Mattin, and J. Lancaster. "Remote sensing techniques in the analysis of change detection". *Geocarto International*, 8:39–50, 1993.

[15] W. K. Pratt. "Correlation techniques of image registration". *IEEE Transactions on Aerospatial Electronic Systems*, 10(1):353–358, 1974.

[16] W. K. Pratt. *Digital image processing*. John Wiley & Sons, primera edition, 1991.

[17] S. A. Sader and J. C. Winne. "RGB-NDVI colour composites for visualizing forest change dynamics". *International Journal of Remote Sensing*, 13:3055–3067, 1992.

[18] J. Shi and Carlo Tomasi. "Good features to track". *IEEE Conference on Computer Vision and Pattern Recognition*, 1994.

[19] G. C. Stockman, S. Kopstein, and S. Bennet. "Matching images to models for registration and object detection via clustering". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 4(1):229–241, 1982.

[20] Carlo Tomasi and Takeo Kanade. "Detection and tracking of point features". *Carnegie Mellon University Technical Report CMU-CS-91-132*, 1991.

[21] P. Van Wie and M. Stein. "A Landsat digital image rectification system". *IEEE Transactions on Geoscience and Electronics*, 15(1):130–137, 1977.

[22] R. Y. Wong and E. L. Hall. "Sequential hierarchical scene matching". *IEEE Transactions on Computers*, 27(1):359–366, 1978.

[23] D. Yuan and C. D. Elvidge. "Comparison of relative radiometric normalization techniques". *ISPRS J. Photogramm. Remote Sensing*, 51:117–126, 1996.

Parte VI

Apéndices

Apéndice A

Código

A.1 Introducción

El siguiente apéndice incluye el código correspondiente a los siguientes componentes software:

- Clases
- Módulos
- Interfaz gráfico

En la figura A.1 podemos observar las relaciones entre distintas clases y las librerías de funciones utilizadas.

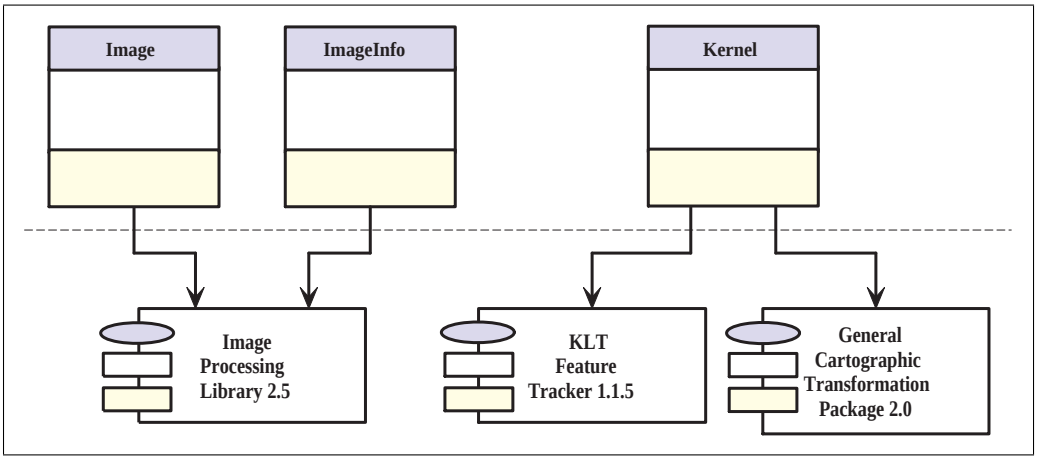


Figura A.1: Componentes software.

A.2 Clases

La clases que encapsulan los componentes IPL (Image Processing Library), KLT (Kanade-Lucas-Tomasi Feature Tracker 1.1.5) y GCTP (General Cartographic Transformation Package 2.0), además de proporcionar funciones adicionales (gestión de disco, parches, etc.) son las siguientes:

1. `Kernel`
2. `Image` (Clase abstracta)
   (a) `DImage`
   (b) `MImage`
3. `ImageInfo`
4. `Polygon`
5. `Matrix<Tipo numérico>`
6. `FastFormat` (Clase estática)

A.2.1 Relación de ficheros

Los ficheros de código asociados a las clases relacionadas anteriormente son los siguientes:

1. `Kernel`
   - `Kernel.h`: Interfaz de Kernel
   - `Kernel.cpp`: Implementación de Kernel
2. `Image` (Clase abstracta)
   - `Image.h`: Interfaz de Image
   - `Image.cpp`: Implementación de Image

   (a) `DImage`
      - `DImage.h`: Interfaz de DImage

- `DImage.cpp`: Implementación de DImage

(b) `MImage`

- `MImage.h`: Interfaz de MImage
- `MImage.cpp`: Implementación de MImage

3. `ImageInfo`
   - `ImageInfo.h`: Interfaz de ImageInfo
   - `ImageInfo.cpp`: Implementación de ImageInfo
4. `Polygon`
   - `Polygon.h`: Interfaz de Polygon
   - `Polygon.cpp`: Implementación de Polygon
5. `Matrix<Tipo numérico>`
   - `Matrix.h`: Interfaz de Matrix
   - `Matrix.cpp`: Implementación de Matrix
6. `FastFormat` (Clase estática)
   - `FastFormat.h`: Interfaz de FastFormat
   - `FastFormat.cpp`: Implementación de FastFormat
7. `IPSoft.h`
   Fichero de cabecera de la librería de clases (ver figura A.2).

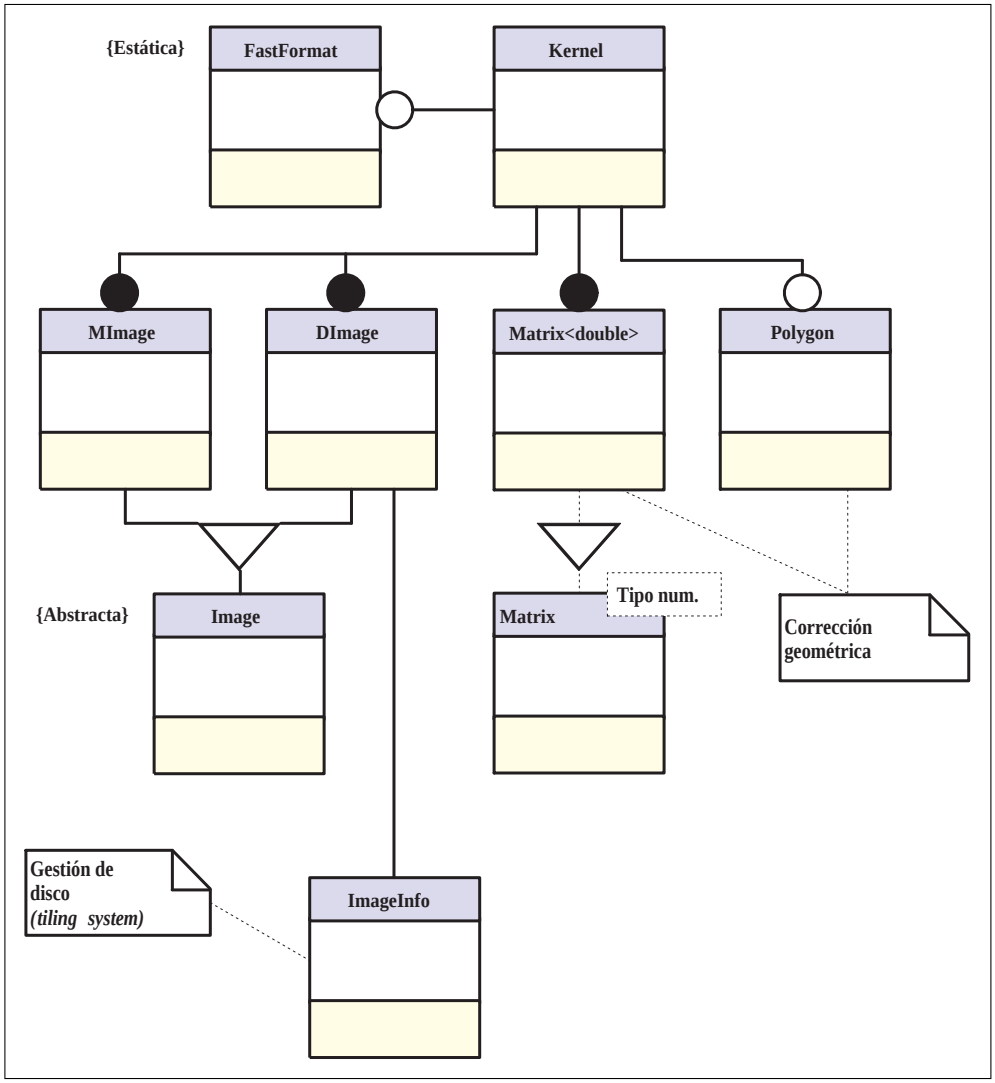


Figura A.2: DC del sistema.

Kernel.h

```
// Kernel.h: interface for the CKernel class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_KERNEL_H_INCLUDED_)
#define AFX_KERNEL_H_INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

//  Classes
```

```
class CImage;
class CMImage;
class CDImage;

class CPolygon;

class CFastFormat;

#include "Matrix.h"     //  Matrix template

class CKernel
{
public:
    CKernel();
    virtual ~CKernel();

//  User functions
//
    bool GetLog();
    void SetLog(bool bLog);

    int GetTileWidth();
    void SetTileWidth(int iWidth);
    int GetTileHeight();
    void SetTileHeight(int iHeight);

    void SetPolygonFile(const char* pFileName);

//  Radiometric correction
//
    void RadiometricCorrection(CImage* pImageR, CImage* pImageC, CImage* pImageX);

//  Geometric correction
//
    void GeometricCorrection(CImage* pImageR, CImage* pImageC, CImage* pImageX);

//  Changes detection
//
    enum { Absolute = 0, Offset = 1 };

    int GetMode();
    void SetMode(int iMode);

    int GetOffset();
    void SetOffset(int iOffset);

    void ChangesDetection(CImage* pImageR, CImage* pImageC, CImage* pImageX);

private:

//  Header file
    CFastFormat* m_pHeaderFile;

//  Log file
    fstream* m_pLogFile;

    bool m_bLog;

//  Tile size
    int m_iTileWidth;
    int m_iTileHeight;
```

```
//  Polygon of interest
    CPolygon* m_pPolygon;

//  Changes detection
//
    int m_iOffset;  //  Offset
    int m_iMode;    //  Mode

    CDMatrix Coefficients(CDMatrix R, CDMatrix C);
    int Residuals(CDMatrix R, CDMatrix C, CDMatrix X);
};

#endif // !defined(AFX_KERNEL_H_INCLUDED_)
```

Kernel.cpp

```
// Kernel.cpp: implementation of the CKernel class.
//
//////////////////////////////////////////////////////////////////////

#include "IPSoft.h"

//////////////////////////////////////////////////////////////////////
// Construction/Destruction

/*
//////////////////////////////////////////////////////////////////////
// Name:        CKernel
// Purpose:     Constructor
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
// Notes:
*/
CKernel::CKernel()
{
//  Log file flag
    m_bLog = false;

//  Log file object
    m_pLogFile = new fstream("DetCam.log", ios::out);

//  Geometric correction (default)
//
    m_iTileWidth = 128;
    m_iTileHeight = 128;

//  Polygon
    m_pPolygon = NULL;

//  Changes detection (default)
//
    m_iOffset = 127;
    m_iMode = Absolute;
}

/*
//////////////////////////////////////////////////////////////////////
// Name:        ~CKernel
// Purpose:     Destructor
// Context:     i42aresv Soft
```

```
// Returns:     void
// Parameters:
// Notes:
*/
CKernel::~CKernel()
{
//  Close log file
    m_pLogFile->close();

//  Remove log file
    if(m_bLog == false)
        remove("DetCam.log");

//  Delete log file object
    delete m_pLogFile;

//  Delete polygon object
    if(m_pPolygon != NULL)
        delete m_pPolygon;
}

////////////////////////////////////////////////////////////////////////
// Operations

/*
////////////////////////////////////////////////////////////////////////
// Name:        GetLog
// Purpose:     Gets log status
// Context:     i42aresv Soft
// Returns:     bool - true: Enable / false: Disable
// Parameters:
// Notes:
*/
bool CKernel::GetLog()
{
    return m_bLog;
}

/*
////////////////////////////////////////////////////////////////////////
// Name:        SetLog
// Purpose:     Sets log status
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
        bool bLog - Log status
// Notes:
*/
void CKernel::SetLog(bool bLog)
{
    m_bLog = bLog;
}

////////////////////////////////////////////////////////////////////////
// Radiometric correction

/*
////////////////////////////////////////////////////////////////////////
// Name:        RadiometricCorrection
// Purpose:     Performs the radiometric correction
// Context:     i42aresv Soft
// Returns:     void
```

```
// Parameters:
        CImage* pImageR - Reference image
        CImage* pImageC - Image to correct
        CImage* pImageX - Resultant image
// Notes:
*/
void CKernel::RadiometricCorrection(CImage* pImageR,
        CImage* pImageC, CImage* pImageX)
{
//  Process status
    TRACE("\nRadiometric correction\nbegin\n")

//  Grey levels
    int iLevels = CImage::m_iLevels;

//  Process status
    TRACE("\n\tEqualize histogram ...")

//  Equalize histogram for image to correct
    long* pHistogramC = new long[iLevels];
    pImageC->EqualizeHistogram(pImageX, pHistogramC);

//  Process status
    TRACE("... done\n")
    TRACE("\n\tCompute histogram ...")

//  Compute histogram for reference image
    long* pHistogramR = new long[iLevels];
    pImageR->ComputeHistogram(pHistogramR);

//  Process status
    TRACE("... done\n")

//  Transformation functions
    long* T = new long[iLevels];
    long* G = new long[iLevels];

//  Counters
    register int i;
    register int j;

//  Process status
    TRACE("\n\tCompute Inv(G) function ...")

//  Compute G and T functions
    T[0] = pHistogramC[0];
    G[0] = pHistogramR[0];
    for(i = 1; i < iLevels; i++)
    {
        T[i] = T[i - 1] + pHistogramC[i];
        G[i] = G[i - 1] + pHistogramR[i];
    }

//  Specified histogram for image to correct
    long* InvG = new long[iLevels];

//  Compute Inv(G) function
    long iMinValue;

    for(i = 0; i < iLevels; i++)
    {
        iMinValue = abs(T[i] - G[0]);
```

```
            InvG[i] = 0;
            for(j = 1; j < iLevels; j++)
            {
                if(abs(T[i] - G[j]) < iMinValue)
                {
                    iMinValue = abs(T[i] - G[j]);
                    InvG[i] = j;
                }
            }
        }

    //  Process status
        TRACE("... done\n")

        if(m_bLog == true)
        {
        //  Log buffer
            char pBuffer[128];

        //  Head
            sprintf(pBuffer, "\n// Radiometric correction");
            m_pLogFile->write(pBuffer, strlen(pBuffer));
            sprintf(pBuffer, "\n// **********************");
            m_pLogFile->write(pBuffer, strlen(pBuffer));
            sprintf(pBuffer, "\n// begin");

        //  Content
            m_pLogFile->write(pBuffer, strlen(pBuffer));
            sprintf(pBuffer, "\n[Radiometric correction]");
            m_pLogFile->write(pBuffer, strlen(pBuffer));
            sprintf(pBuffer, "\n[Inv(G) function]");
            m_pLogFile->write(pBuffer, strlen(pBuffer));
            for(i = 0; i < iLevels; i++)
            {
                sprintf(pBuffer, "\n%d", InvG[i]);
                m_pLogFile->write(pBuffer, strlen(pBuffer));
            }

        //  Tail
            sprintf(pBuffer, "\n//\tend\n");
            m_pLogFile->write(pBuffer, strlen(pBuffer));
        }

    //  Process status
        TRACE("\n\tCorrect image ...")

    //  Histogram specification
        pImageX->ContrastStretch(InvG); //  IMPORTANT: X image

    //  Process status
        TRACE("... done\n")

    //  Delete objects
        delete [] G;
        delete [] T;
        delete [] InvG;

        delete [] pHistogramR;
        delete [] pHistogramC;

    //  Process status
        TRACE("\nend\n")
```

```
}

//////////////////////////////////////////////////////////////////////
// Geometric correction

/*
//////////////////////////////////////////////////////////////////////
// Name:        GetTileWidth
// Purpose:     Gets tile width
// Context:     i42aresv Soft
// Returns:     int - Tile width
// Parameters:
// Notes:
*/
int CKernel::GetTileWidth()
{
    return m_iTileWidth;
}

/*
//////////////////////////////////////////////////////////////////////
// Name:        SetTileWidth
// Purpose:     Sets tile width
// Context:     i42aresv Soft
// Returns:
// Parameters:
        int iWidth - Tile width
// Notes:
*/
void CKernel::SetTileWidth(int iWidth)
{
    m_iTileWidth = iWidth;
}

/*
//////////////////////////////////////////////////////////////////////
// Name:        GetTileHeight
// Purpose:     Gets tile height
// Context:     i42aresv Soft
// Returns:     int - Tile height
// Parameters:
// Notes:
*/
int CKernel::GetTileHeight()
{
    return m_iTileHeight;
}

/*
//////////////////////////////////////////////////////////////////////
// Name:        SetTileHeight
// Purpose:     Sets tile height
// Context:     i42aresv Soft
// Returns:
// Parameters:
        int iHeight - Tile height
// Notes:
*/
void CKernel::SetTileHeight(int iHeight)
{
    m_iTileHeight = iHeight;
}
```

```
/*
////////////////////////////////////////////////////////////////////////
// Name:        SetPolygonFile
// Purpose:     Sets polygon file
// Context:     i42aresv Soft
// Returns:
// Parameters:
        const char* pFileName - Polygon file name
// Notes:
*/
void CKernel::SetPolygonFile(const char* pFileName)
{
    if(_access(pFileName, 0) != -1)
    {
    //  Polygon file
        ifstream fFile(pFileName);

    //  Buffer
        char pBuffer[128];

    //  Number of vertexes
        int iSize;

        if(fFile.eof() == FALSE)
        {
            fFile.getline(pBuffer, 128);
            sscanf(pBuffer,"%d", &iSize);
        }

        POINT* pPolygon = new POINT[iSize];

        register int i = 0;
        while(fFile.eof() == FALSE)
        {
            fFile.getline(pBuffer, 128);
            sscanf(pBuffer,"%d\t%d", &(pPolygon[i].x), &(pPolygon[i].y));
            i++;    //  Next vertex
        }

        m_pPolygon = new CPolygon(iSize, pPolygon);

        delete [] pPolygon;
    }
}

/*
////////////////////////////////////////////////////////////////////////
// Name:        Coefficients
// Purpose:     Computes affine/bilinear transform coefficients
// Context:     i42aresv Soft
// Returns:
// Parameters:
        CDMatrix R - GCPs coordinates (Reference image)
        CDMatrix C - GCPs coordinates (Image to correct)
// Notes:
*/
CDMatrix CKernel::Coefficients(CDMatrix R, CDMatrix C)
{
    register int i; //  Cols
    register int j; //  Rows
    register int k; //  Points
```

```
    long iNPoints = R.GetCols();        //  C.GetCols();
    long iNTerms = 4;   //  x' = a0 + a1 * x + a2 * y + a3 * x * y
                        //  y' = b0 + b1 * x + b2 * y + b3 * x * y

    CDMatrix D(iNTerms, iNPoints);  //  Data matrix
    CDMatrix V(iNTerms, iNTerms);   //  Variances matrix (square)

    CDMatrix X(2, iNTerms); //  Coefficients matrix

    double* M = new double[iNTerms];    //  Means vector

//  Data
    for(k = 0; k < iNPoints; k++)
    {
        D(1, k) = C(0, k);  //  x
        D(2, k) = C(1, k);  //  y
        D(3, k) = (C(0, k) * C(1, k));  //  x * y
    }

//  Means
    for(j = 1; j < iNTerms; j++)
    {
        M[j] = 0.0;
        for(k = 0; k < iNPoints; k++)
            M[j] += D(j, k);
        M[j] /= iNPoints;
    }

//  Variances
    for(j = 1; j < iNTerms; j++)
        for(i = 1; i < iNTerms; i++)
        {
            V(j, i) = 0.0;
            for(k = 0; k < iNPoints; k++)
                V(j, i) += ((D(j, k) - M[j]) * (D(i, k) - M[i]));
            V(j, i) /= iNPoints;
        }

    register int l;
    for(l = 0; l < 2; l++)
    {
    //  Data
        for(k = 0; k < iNPoints; k++)
            D(0, k) = R(l, k);

    //  Means
        M[0] = 0.0;
        for(k = 0; k < iNPoints; k++)
            M[0] += D(0, k);
        M[0] /= iNPoints;

    //  Variances
        V(0, 0) = 0.0;
        for(k = 0; k < iNPoints; k++)
            V(0, 0) += ((D(0, k) - M[0]) * (D(0, k) - M[0]));
        V(0, 0) /= iNPoints;

        for(j = 0; j < iNTerms; j++)
        {
            V(j, 0) = 0.0;
            for(k = 0; k < iNPoints; k++)
```

```
                V(j, 0) += ((D(j, k) - M[j]) * (D(0, k) - M[0]));
            V(j, 0) /= iNPoints;
        }

        for(i = 0; i < iNTerms; i++)
        {
            V(0, i) = 0.0;
            for(k = 0; k < iNPoints; k++)
                V(0, i) += ((D(0, k) - M[0]) * (D(i, k) - M[i]));
            V(0, i) /= iNPoints;
        }

    //  Coefficients (a1, aN)
        for(i = 1; i < iNTerms; i++)
            X(l, i) = -(pow(-1, i) * V.SubMatrix(0, i).Determinant() /
                            V.SubMatrix(0, 0).Determinant());

    //  Coefficient a0
        X(l, 0) = M[0];
        for(i = 1; i < iNTerms; i++)
            X(l, 0) -= (X(l, i) * M[i]);
    }

//  Coefficients
    return X;
}

/*
//////////////////////////////////////////////////////////////////////
// Name:        Residuals
// Purpose:     Computes GCPs residuals and selects the worst point
// Context:     i42aresv Soft
// Returns:
// Parameters:
        CDMatrix R - GCPs coordinates (Reference image)
        CDMatrix C - GCPs coordinates (Image to correct)
        CDMatrix X - Coefficients matrix
// Notes:
*/
int CKernel::Residuals(CDMatrix R, CDMatrix C, CDMatrix X)
{
    int iNPoints = R.GetCols(); //  C.GetCols();

    double y;   //  x value
    double x;   //  y value

    double EL;  //  Residual
    double RMS = 0.0;

    double dMax = 0.0;  //  Residual with max value (candidate point)
    int iMax = -1;  //  Candidate point

    register int k;
    for(k = 0; k < iNPoints; k++)
    {
        x = X(0, 0) + (X(0, 1) * C(0, k)) +
            (X(0, 2) * C(1, k)) + (X(0, 3) * C(0, k) * C(1, k));
        y = X(1, 0) + (X(1, 1) * C(0, k)) +
            (X(1, 2) * C(1, k)) + (X(1, 3) * C(0, k) * C(1, k));

        EL = sqrt((pow((R(0, k) - x), 2) + pow((R(1, k) - y), 2)));
```

```
    //  Select the worst point (candidate point)
        if((EL > 1.0) && (EL > dMax))
        {
            iMax = k;
            dMax = EL;
        }

        RMS += (pow((R(0, k) - x), 2) + pow((R(1, k) - y), 2));
    }
//  Compute root mean squared
    RMS /= iNPoints;

    if(sqrt(RMS) > 1.0)
        return iMax;    //  The worst point
    else
        return -1;
}

/*
//////////////////////////////////////////////////////////////////////
// Name:        GeometricCorrection
// Purpose:     Performs the geometric correction
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
        CImage* pImageR - Reference image
        CImage* pImageC - Image to correct
        CImage* pImageX - Resultant image
// Notes:
*/
void CKernel::GeometricCorrection(CImage* pImageR,
        CImage* pImageC, CImage* pImageX)
{
//  Process status
    TRACE("\nGeometric correction\nbegin\n")

//  Number of tiles
    int iXTiles = (pImageR->GetWidth() / m_iTileWidth); //  Horizontal
    int iYTiles = (pImageR->GetHeight() / m_iTileHeight);   //  Vertical

    int iFeatures = 25;

//  KLT environment
    KLT_TrackingContext tc = KLTCreateTrackingContext();
    tc->window_width  = 15;
    tc->window_height = 15;
    KLTChangeTCPyramid(tc, 15);
    KLTUpdateTCBorder(tc);

    KLT_FeatureList flR = KLTCreateFeatureList(iFeatures);
    KLT_FeatureList flC = KLTCreateFeatureList(iFeatures);

    KLTSetVerbosity(0);

//  Tiles
    CMImage TileR(m_iTileWidth, m_iTileHeight);
    CMImage TileC(m_iTileWidth, m_iTileHeight);

    CDMatrix R(2, (iXTiles * iYTiles));
    CDMatrix C(2, (iXTiles * iYTiles));
    CDMatrix X(2, m_iNTerms);
```

```
//  Counters
    register int i;
    register int xt;
    register int yt;

//  Process status
    TRACE("\n\tCompute GPCs ...")

//  Create regions of interest
    pImageR->CreateROI(
        0, 0,
        m_iTileWidth, m_iTileHeight);
    pImageC->CreateROI(
        0, 0,
        m_iTileWidth, m_iTileHeight);

//  Rows
    for(yt = 0; yt < iYTiles; yt++)
    //  Cols
        for(xt = 0; xt < iXTiles; xt++)
        {
        //  Polygon of interest (POI) ?
            if(m_pPolygon != NULL)
            {
            //  Tile inside of POI ?
                if(m_pPolygon->RectInPolygon(
                    (xt * m_iTileWidth), (yt * m_iTileHeight),
                    m_iTileWidth, m_iTileHeight) == false)
                {
                    R(0, (yt * iXTiles + xt)) = -2.0;
                    R(1, (yt * iXTiles + xt)) = -2.0;
                    C(0, (yt * iXTiles + xt)) = -2.0;
                    C(1, (yt * iXTiles + xt)) = -2.0;

                    continue;   //  Next tile
                }
            }

            R(0, (yt * iXTiles + xt)) = -1.0;
            R(1, (yt * iXTiles + xt)) = -1.0;
            C(0, (yt * iXTiles + xt)) = -1.0;
            C(1, (yt * iXTiles + xt)) = -1.0;

        //  Set regions of interest
            pImageR->SetROI(
                (xt * m_iTileWidth), (yt * m_iTileHeight),
                m_iTileWidth, m_iTileHeight);
            pImageC->SetROI(
                (xt * m_iTileWidth), (yt * m_iTileHeight),
                m_iTileWidth, m_iTileHeight);

        //  Tile from reference image
            TileR.Copy(pImageR);

        //  Tile from image to correct
            TileC.Copy(pImageC);

            KLTSelectGoodFeatures(  //  Select the 25 best features
                tc,
                TileR.GetImageData(),
                m_iTileWidth, m_iTileHeight,
                flC);
```

```
            for(i = 0; i < iFeatures; i++)
            {
                flR->feature[i]->x = flC->feature[i]->x;
                flR->feature[i]->y = flC->feature[i]->y;
                flR->feature[i]->val = flC->feature[i]->val;
            }

            KLTTrackFeatures(   //  Track the 25 best features
                tc,
                TileR.GetImageData(), TileC.GetImageData(),
                m_iTileWidth, m_iTileHeight,
                flC);

            i = 0;
            do
            {
                if((flC->feature[i]->x > -1) && (flC->feature[i]->y > -1))
                {
                //  Reference image
                    R(0, (yt * iXTiles + xt)) =
                            (flR->feature[i]->x + (xt * m_iTileWidth));
                    R(1, (yt * iXTiles + xt)) =
                            (flR->feature[i]->y + (yt * m_iTileHeight));

                //  Image to correct
                    C(0, (yt * iXTiles + xt)) =
                            (flC->feature[i]->x + (xt * m_iTileWidth));
                    C(1, (yt * iXTiles + xt)) =
                            (flC->feature[i]->y + (yt * m_iTileHeight));
                    i = iFeatures;
                }
                i++;
            }
            while(i < iFeatures);
        }

//  Delete regions of interest
    pImageR->DeleteROI();
    pImageC->DeleteROI();

//  Process status
    TRACE("... done\n");

    if(m_bLog == true)
    {
    //  Log buffer
        char pBuffer[128];

    //  Head
        sprintf(pBuffer, "\n// Geometric correction");
        m_pLogFile->write(pBuffer, strlen(pBuffer));
        sprintf(pBuffer, "\n// ********************");
        m_pLogFile->write(pBuffer, strlen(pBuffer));
        sprintf(pBuffer, "\n// begin");

    //  Content
        m_pLogFile->write(pBuffer, strlen(pBuffer));
        sprintf(pBuffer, "\n[Geometric correction]");
        m_pLogFile->write(pBuffer, strlen(pBuffer));
        sprintf(pBuffer, "\n[Tile width]\n%d", m_iTileWidth);
        m_pLogFile->write(pBuffer, strlen(pBuffer));
```

```
        sprintf(pBuffer, "\n[Tile height]\n%d", m_iTileHeight);
        m_pLogFile->write(pBuffer, strlen(pBuffer));
        sprintf(pBuffer, "\n[X tile number]\n%d", iXTiles);
        m_pLogFile->write(pBuffer, strlen(pBuffer));
        sprintf(pBuffer, "\n[Y tile number]\n%d", iYTiles);
        m_pLogFile->write(pBuffer, strlen(pBuffer));
        sprintf(pBuffer, "\n[Tile number]\n%d", (iXTiles * iYTiles));
        m_pLogFile->write(pBuffer, strlen(pBuffer));

        sprintf(pBuffer, "\n// KLT algorithm");
        m_pLogFile->write(pBuffer, strlen(pBuffer));
        sprintf(pBuffer, "\n[Number of features]\n%d", iFeatures);
        m_pLogFile->write(pBuffer, strlen(pBuffer));
        sprintf(pBuffer, "\n[Window width]\n%d", tc->window_width);
        m_pLogFile->write(pBuffer, strlen(pBuffer));
        sprintf(pBuffer, "\n[Window height]\n%d", tc->window_height);
        m_pLogFile->write(pBuffer, strlen(pBuffer));

        sprintf(pBuffer, "\n// -1: Bad tile; -2: Tile out of polygon");
        m_pLogFile->write(pBuffer, strlen(pBuffer));
        sprintf(pBuffer, "\n// Tile.y, Tile.x, R.x, R.y, C.x, C.y, CX, CY");
        m_pLogFile->write(pBuffer, strlen(pBuffer));
        sprintf(pBuffer, "\n[Tracked points]");
        m_pLogFile->write(pBuffer, strlen(pBuffer));

    //  Offsets
        double cx, cy;

        i = 0;
    //  Rows
        for(yt = 0; yt < iYTiles; yt++)
        //  Cols
            for(xt = 0; xt < iXTiles; xt++)
            {
                cx = (R(0, (yt * iXTiles + xt)) - C(0, (yt * iXTiles + xt)));
                cy = (R(1, (yt * iXTiles + xt)) - C(1, (yt * iXTiles + xt)));
                sprintf(pBuffer, "\n%d\t%d\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf",
                    yt, xt,
                    R(0, (yt * iXTiles + xt)), R(1, (yt * iXTiles + xt)),
                    C(0, (yt * iXTiles + xt)), C(1, (yt * iXTiles + xt)),
                    cx, cy);
                m_pLogFile->write(pBuffer, strlen(pBuffer));

                if( (R(0, (yt * iXTiles + xt)) > -1.0) &&
                    (R(1, (yt * iXTiles + xt)) > -1.0) &&
                    (C(0, (yt * iXTiles + xt)) > -1.0) &&
                    (C(1, (yt * iXTiles + xt)) > -1.0))
                    i++;
            }

        sprintf(pBuffer, "\n[Number of tracked points]\n%d", i);
        m_pLogFile->write(pBuffer, strlen(pBuffer));
        sprintf(pBuffer, "\n// Tile.y, Tile.x, R.x, R.y, C.x, C.y, CX, CY");
        m_pLogFile->write(pBuffer, strlen(pBuffer));
        sprintf(pBuffer, "\n[Deleted points]");
        m_pLogFile->write(pBuffer, strlen(pBuffer));
    }

//  Process status
    TRACE("\n\tRemove bad GPCs ...")

//  Rows
```

```
    for(yt = (iYTiles - 1); yt > -1; yt--)
    //  Cols
        for(xt = (iXTiles - 1); xt > -1; xt--)
            if( (R(0, (yt * iXTiles + xt)) < 0.0) &&
                (R(1, (yt * iXTiles + xt)) < 0.0) &&
                (C(0, (yt * iXTiles + xt)) < 0.0) &&
                (C(1, (yt * iXTiles + xt)) < 0.0))
            {
                R.RemoveCol((yt * iXTiles + xt));
                C.RemoveCol((yt * iXTiles + xt));
            }

//  Process status
    TRACE("... done\n")
    TRACE("\n\tCompute coefficients ...")

    int iPoint;
    do
    {
        X = Coefficients(R, C);
        iPoint = Residuals(R, C, X);
        if(iPoint > -1)
        {
            if(m_bLog == true)
            {
            //  Log buffer
                char pBuffer[128];

                sprintf(pBuffer, "\n%d\t%d\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf",
                    ((int)(R(1, iPoint) + 0.5) / m_iTileHeight),
                    ((int)(R(0, iPoint) + 0.5) / m_iTileWidth),
                    R(0, iPoint), R(1, iPoint),
                    C(0, iPoint), C(1, iPoint),
                    (R(0, iPoint) - C(0, iPoint)),
                    (R(1, iPoint) - C(1, iPoint)));
                m_pLogFile->write(pBuffer, strlen(pBuffer));
            }

        //  Remove candidate point
            R.RemoveCol(iPoint);
            C.RemoveCol(iPoint);
        }
    }
    while(iPoint > -1);

//  Process status
    TRACE("... done\n")

    if(m_bLog == true)
    {
    //  Log buffer
        char pBuffer[128];

    //  Content
        sprintf(pBuffer, "\n[End]");
        m_pLogFile->write(pBuffer, strlen(pBuffer));

        sprintf(pBuffer, "\n[Number of deleted points]\n%d", (i - R.GetCols()));
        m_pLogFile->write(pBuffer, strlen(pBuffer));

        sprintf(pBuffer, "\n// x' = a0 + a1 * x + a2 * y + a3 * x * y");
        m_pLogFile->write(pBuffer, strlen(pBuffer));
```

```
            sprintf(pBuffer, "\n// y' = b0 + b1 * x + b2 * y + b3 * x * y");
            m_pLogFile->write(pBuffer, strlen(pBuffer));

            sprintf(pBuffer, "\n[Coefficients]");
            m_pLogFile->write(pBuffer, strlen(pBuffer));

            sprintf(pBuffer, "\n%lf\t%lf\t%lf\t%lf",
                X(0, 0), X(0, 1), X(0, 2), X(0, 3));
            m_pLogFile->write(pBuffer, strlen(pBuffer));

            sprintf(pBuffer, "\n%lf\t%lf\t%lf\t%lf",
                X(1, 0), X(1, 1), X(1, 2), X(1, 3));
            m_pLogFile->write(pBuffer, strlen(pBuffer));

        //  Tail
            sprintf(pBuffer, "\n//\tend\n");
            m_pLogFile->write(pBuffer, strlen(pBuffer));
        }

    //  Process status
        TRACE("\n\tCorrect image ...")

    //  Warp bilinear function
        pImageC->WarpBilinear(pImageX, X);

    //  Free feature lists
        KLTFreeFeatureList(flR);
        KLTFreeFeatureList(flC);

    //  Free tracking context
        KLTFreeTrackingContext(tc);

    //  Process status
        TRACE("... done\n")

    //  Process status
        TRACE("\nend\n")
    }

    ////////////////////////////////////////////////////////////////////////
    // Changes detection

    /*
    ////////////////////////////////////////////////////////////////////////
    // Name:        GetMode
    // Purpose:     Gets mode
    // Context:     i42aresv Soft
    // Returns:     int - Change detection mode (Absolute/Offset)
    // Parameters:
    // Notes:
    */
    int CKernel::GetMode()
    {
        return m_iMode;
    }

    /*
    ////////////////////////////////////////////////////////////////////////
    // Name:        SetMode
    // Purpose:     Sets mode
    // Context:     i42aresv Soft
    // Returns:     void
```

```
// Parameters:
        int iMode - Mode
// Notes:
*/
void CKernel::SetMode(int iMode)
{
    m_iMode = iMode;
}

/*
/////////////////////////////////////////////////////////////////////
// Name:        GetOffset
// Purpose:     Gets offset
// Context:     i42aresv Soft
// Returns:     int - Offset
// Parameters:
// Notes:
*/
int CKernel::GetOffset()
{
    return m_iOffset;
}

/*
/////////////////////////////////////////////////////////////////////
// Name:        SetOffset
// Purpose:     Sets offset
// Context:     i42aresv Soft
// Returns:
// Parameters:
        int iOffset - Offset
// Notes:
*/
void CKernel::SetOffset(int iOffset)
{
    m_iOffset = iOffset;
}

/*
/////////////////////////////////////////////////////////////////////
// Name:        ChangesDetection
// Purpose:     Detects changes in source images
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
        CImage* pImageR - Reference image
        CImage* pImageC - Image to correct
        CImage* pImageX - Resultant image
// Notes:
*/
void CKernel::ChangesDetection(CImage* pImageR,
        CImage* pImageC, CImage* pImageX)
{
//  Process status
    TRACE("\nChanges detection\nbegin\n")

//  Process status
    TRACE("\n\tDetect changes ...")

    switch(m_iMode)
    {
    case Absolute:
```

```
            pImageR->Subtract_Abs(pImageC, pImageX);
            break;
        case Offset:
            pImageR->Subtract_AddScalar(pImageC, m_iOffset, pImageX);
            break;
        default:
            cerr << "ERROR <ChangesDetection>: (iMode != Abs) && \
                (iMode != Offset)" << endl;
            exit(-1);
        }

    //  Process status
        TRACE("... done\n");

        if(m_bLog == true)
        {
        //  Log buffer
            char pBuffer[128];

        //  Head
            sprintf(pBuffer, "\n// Changes detection");
            m_pLogFile->write(pBuffer, strlen(pBuffer));
            sprintf(pBuffer, "\n// *****************");
            m_pLogFile->write(pBuffer, strlen(pBuffer));
            sprintf(pBuffer, "\n// begin");

        //  Content
            m_pLogFile->write(pBuffer, strlen(pBuffer));
            sprintf(pBuffer, "\n[Changes detection]");
            m_pLogFile->write(pBuffer, strlen(pBuffer));
            sprintf(pBuffer, "\n// 0: DNd = abs(DNr - DNc)");
            m_pLogFile->write(pBuffer, strlen(pBuffer));
            sprintf(pBuffer, "\n// 1: DNd = (DNr - DNc + Offset)");
            m_pLogFile->write(pBuffer, strlen(pBuffer));
            sprintf(pBuffer, "\n[Mode]");
            m_pLogFile->write(pBuffer, strlen(pBuffer));
            if(m_iMode == Absolute)
                sprintf(pBuffer, "\n0");
            else
                sprintf(pBuffer, "\n1");
            m_pLogFile->write(pBuffer, strlen(pBuffer));

        //  Tail
            sprintf(pBuffer, "\n//\tend\n");
            m_pLogFile->write(pBuffer, strlen(pBuffer));
        }

    //  Process status
        TRACE("\nend\n")
    }
```

Image.h

```
    // Image.h: interface for the CImage class.
    //
    //////////////////////////////////////////////////////////////////////

    #if !defined(AFX_IMAGE_H_INCLUDED_)
    #define AFX_IMAGE_H_INCLUDED_

    #if _MSC_VER > 1000
```

```
#pragma once
#endif // _MSC_VER > 1000

//  Classes
class CMImage;

class CImage
{
public:
    CImage(int iWidth, int iHeight);
    virtual ~CImage();

//  Data exchange functions
//
    void Copy(CImage* pImage);

//  Dyadic arithmetic operations
//
    virtual void Subtract_Abs(CImage* pImageC, CImage* pImageX);
    virtual void Subtract_AddScalar(CImage* pImageC, BYTE byScalar,
            CImage* pImageX);

//  Histogram functions
//

    virtual void ComputeHistogram(long* pHistogram);
    virtual void ContrastStretch(long* pHistogram);

    virtual void EqualizeHistogram(CImage* pImage, long* pHistogram);

//  Image geometric transform functions
//
    virtual void WarpAffine(CImage* pImage, double dCoeffs[2][3]);
    virtual void WarpBilinear(CImage* pImage, double dCoeffs[2][4]);

//  User functions
//
    virtual int LoadRAWFile(const char* pFileName) = 0;
    virtual void SaveRAWFile(const char* pFileName) = 0;

//  For Tcl/Tk interface
    virtual void SavePPMFile(const char* pFileName) = 0;

    int GetWidth();
    int GetHeight();

    void CreateROI(int iXOffset, int iYOffset, int iWidth, int iHeight);
    void SetROI(int iXOffset, int iYOffset, int iWidth, int iHeight);
    void DeleteROI();

//  Grey levels
    static const int m_iLevels;

//  Tile size
    static const int m_iTileWidth;
    static const int m_iTileHeight;

protected:

    IplImage* m_pImage;
};
```

```
#endif // !defined(AFX_IMAGE_H_INCLUDED_)
```

Image.cpp

```
// Image.cpp: implementation of the CImage class.
//
//////////////////////////////////////////////////////////////////////

#include "IPSoft.h"

//  Grey levels
const int CImage::m_iLevels = 256;

//  Tile size
const int CImage::m_iTileWidth  = 128;
const int CImage::m_iTileHeight = 128;

//////////////////////////////////////////////////////////////////////
// Construction/Destruction

/*
//////////////////////////////////////////////////////////////////////
// Name:        CImage
// Purpose:     Constructor
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
        int iWidth - Image width
        int iHeight - Image height
// Notes:
*/
CImage::CImage(int iWidth, int iHeight)
{
    m_pImage = iplCreateImageHeader(
        1,
        0,
        IPL_DEPTH_8U,
        "GRAY",
        "GRAY",
        IPL_DATA_ORDER_PIXEL,
        IPL_ORIGIN_TL,
        IPL_ALIGN_DWORD,
        iWidth,
        iHeight,
        NULL,
        NULL,
        NULL,
        NULL);
}

/*
//////////////////////////////////////////////////////////////////////
// Name:        ~CImage
// Purpose:     Destructor
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
// Notes:
*/
CImage::~CImage()
{
```

```
    iplDeallocate(m_pImage, IPL_IMAGE_HEADER);
}

////////////////////////////////////////////////////////////////////
//  Data exchange functions

/*
////////////////////////////////////////////////////////////////////
// Name:        Copy
// Purpose:     Copies image data from source image
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
        CImage* pImage - Source image
// Notes:
*/
void CImage::Copy(CImage* pImage)
{
    iplCopy(pImage->m_pImage, m_pImage);
}

////////////////////////////////////////////////////////////////////
//  Image geometric transform functions

/*
////////////////////////////////////////////////////////////////////
// Name:        WarpAffine
// Purpose:     Performs affine transforms with the specified coefficients
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
        CImage* pImageX - Resultant image
        CDMatrix X - Affine transform coefficients
// Notes:
*/ void CImage::WarpAffine(CImage* pImageX, CDMatrix X) {
//  Counters
    register int i;
    register int j;

//  Coefficients
    double dCoeffs[2][3];

//  Prepare array to iplWarpAffine
    for(j = 0; j < 2; j++)
    {
        dCoeffs[j][2] = X(j, 0);
        for(i = 1; i < 3; i++)
            dCoeffs[j][(i - 1)] = X(j, i);
    }

//  Warp affine
    iplWarpAffine(m_pImage, pImageX->m_pImage, dCoeffs,
        IPL_INTER_CUBIC | IPL_SMOOTH_EDGE);
}

/*
////////////////////////////////////////////////////////////////////
// Name:        WarpBilinear
// Purpose:     Performs a bilinear transform with the specified coefficients
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
```

```
        CImage* pImageX - Resultant image
        CDMatrix X - Bilinear transform coefficients
// Notes:
*/ void CImage::WarpBilinear(CImage* pImageX, CDMatrix X) {
//  Counters
    register int i;
    register int j;

//  Coefficients
    double dCoeffs[2][4];

//  Prepare array to iplWarpBilinear
    for(j = 0; j < 2; j++)
    {
        dCoeffs[j][3] = X(j, 0);
        for(i = 1; i < 3; i++)
            dCoeffs[j][i] = X(j, i);
        dCoeffs[j][0] = X(j, 3);
    }

//  Warp bilinear
    iplWarpBilinear(m_pImage, pImageX->m_pImage, dCoeffs,
        IPL_WARP_R_TO_Q, IPL_INTER_CUBIC | IPL_SMOOTH_EDGE);
}

////////////////////////////////////////////////////////////////////////
//  User functions

/*
////////////////////////////////////////////////////////////////////////
// Name:        GetWidth
// Purpose:     Gets image width
// Context:     i42aresv Soft
// Returns:     int - Image widht
// Parameters:
// Notes:
*/
int CImage::GetWidth()
{
    return m_pImage->width;
}

/*
////////////////////////////////////////////////////////////////////////
// Name:        GetHeight
// Purpose:     Gets image height
// Context:     i42aresv Soft
// Returns:     int - Image height
// Parameters:
// Notes:
*/
int CImage::GetHeight()
{
    return m_pImage->height;
}

/*
////////////////////////////////////////////////////////////////////////
// Name:        CreateROI
// Purpose:     Creates a region of interest(ROI) header with specified attributes
// Context:     i42aresv Soft
// Returns:     void
```

```
// Parameters:
        int x, int y - Offsets from the origin of the rectangular region
        int iWidth, int iHeight - Size of rectangular region
// Notes:
*/
void CImage::CreateROI(int x, int y, int iWidth, int iHeight)
{
    m_pImage->roi = iplCreateROI(0, x, y, iWidth, iHeight);
}

/*
/////////////////////////////////////////////////////////////////////
// Name:        SetROI
// Purpose:     Sets a region of interest (ROI) with specified attributes
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
        int x, int y - Offsets from the origin of the rectangular region
        int iWidth, int iHeight - Size of rectangular region
// Notes:
*/
void CImage::SetROI(int x, int y, int iWidth, int iHeight)
{
    iplSetROI(m_pImage->roi, 0, x, y, iWidth, iHeight);
}

/*
/////////////////////////////////////////////////////////////////////
// Name:        DeleteROI
// Purpose:     Deletes a region of interest (ROI) header
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
// Notes:
*/
void CImage::DeleteROI()
{
    iplDeleteROI(m_pImage->roi);
    m_pImage->roi = NULL;   //  VERY IMPORTANT
}

/////////////////////////////////////////////////////////////////////
//  Histogram functions

/*
/////////////////////////////////////////////////////////////////////
// Name:        ComputeHistogram
// Purpose:     Computes the intensity histogram
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
        long* pHistogram - Histogram information
// Notes:
*/
void CImage::ComputeHistogram(long* pHistogram)
{
//  Number of tiles
    int iXTiles = (GetWidth() / m_iTileWidth);  //  Horizontal
    int iYTiles = (GetHeight() / m_iTileHeight);    //  Vertical

//  DN
    BYTE byPixel;
```

```
//  Counters
    register int xt;    //  Tile
    register int yt;
    register int xp;    //  Pixel
    register int yp;

//  Tile
    CMImage Tile(m_iTileWidth, m_iTileHeight);

//  Clear histogram
    memset(pHistogram, 0, (m_iLevels * sizeof(long)));

//  Create region of interest
    CreateROI(
        0, 0,
        m_iTileWidth, m_iTileHeight);

//  Rows
    for(yt = 0; yt < iYTiles; yt++)
    //  Cols
        for(xt = 0; xt < iXTiles; xt++)
        {
        //  Set region of interest
            SetROI(
                (xt * m_iTileWidth), (yt * m_iTileHeight),
                m_iTileWidth, m_iTileHeight);

        //  Tile from image
            Tile.Copy(this);

        //  Rows
            for(yp = 0; yp < m_iTileHeight; yp++)
            //  Cols
                for(xp = 0; xp < m_iTileWidth; xp++)
                {
                    Tile.GetPixel(xp, yp, &byPixel);
                    pHistogram[byPixel]++;
                }
        }

//  Delete region of interest
    DeleteROI();
}

/*
//////////////////////////////////////////////////////////////////////
// Name:        ContrastStretch
// Purpose:     Stretches the contrast of an image using intensity transformation
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
        long* pHistogram - Intensity transformation
// Notes:
*/
void CImage::ContrastStretch(long* pHistogram)
{
//  Histogram
    long* pHistogramTmp = new long[m_iLevels];

//  Number of tiles
    int iXTiles = (GetWidth() / m_iTileWidth);  //  Horizontal
```

```
    int iYTiles = (GetHeight() / m_iTileHeight);    //  Vertical

//  DN
    BYTE byPixel;

//  Counters
    register int xt;    //  Tile
    register int yt;
    register int xp;    //  Pixel
    register int yp;

//  Tile
    CMImage Tile(m_iTileWidth, m_iTileHeight);

//  Clear histogram
    memset(pHistogramTmp, 0, (m_iLevels * sizeof(long)));

//  Create region of interest
    CreateROI(
        0, 0,
        m_iTileWidth, m_iTileHeight);

//  Rows
    for(yt = 0; yt < iYTiles; yt++)
    //  Cols
        for(xt = 0; xt < iXTiles; xt++)
        {
        //  Set region of interest
            SetROI(
                (xt * m_iTileWidth), (yt * m_iTileHeight),
                m_iTileWidth, m_iTileHeight);

        //  Tile from image
            Tile.Copy(this);

        //  Rows
            for(yp = 0; yp < m_iTileHeight; yp++)
            //  Cols
                for(xp = 0; xp < m_iTileWidth; xp++)
                {
                    Tile.GetPixel(xp, yp, &byPixel);
                    byPixel = (BYTE)pHistogram[byPixel];
                    Tile.PutPixel(xp, yp, &byPixel);

                //  Histogram
                    pHistogramTmp[byPixel]++;
                }

        //  Save tile
            Copy(&Tile);
        }

//  Delete region of interest
    DeleteROI();

//  Copy new histogram
    memcpy(pHistogram, pHistogramTmp, (m_iLevels * sizeof(long)));

//  Delete histogram
    delete [] pHistogramTmp;
}
```

```
/*
////////////////////////////////////////////////////////////////////
// Name:        EqualizeHistogram
// Purpose:     Enhances an image by flattening its intensity histogram
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
        CImage* pImage - Resultant image
        long* pHistogram - Histogram information
// Notes:
*/
void CImage::EqualizeHistogram(CImage* pImage, long* pHistogram)
{
//  F function
    double* F = new double[m_iLevels];

//  T function (transformation function)
    long* T = new long[m_iLevels];

//  Number of tiles
    int iXTiles = (GetWidth() / m_iTileWidth);  //  Horizontal
    int iYTiles = (GetHeight() / m_iTileHeight);    //  Vertical

//  DN
    BYTE byPixel;

//  Counters
    register int i;

    register int xt;    //  Tile
    register int yt;
    register int xp;    //  Pixel
    register int yp;

//  Tile
    CMImage Tile(m_iTileWidth, m_iTileHeight);

//  Clear histogram
    memset(pHistogram, 0, (m_iLevels * sizeof(long)));

//  Compute histogram
    ComputeHistogram(pHistogram);

//  Factor
    double dFactor = (1.0 / (GetWidth() * GetHeight()));

//  Compute F and T function
    F[0] = (pHistogram[0] * dFactor);
    T[0] = (int)(F[0] * (m_iLevels - 1) + 0.5);
    for(i = 1; i < m_iLevels; i++)
    {
        F[i] = F[(i - 1)] + (pHistogram[i] * dFactor);
        T[i] = (int)(F[i] * (m_iLevels - 1) + 0.5);
    }

//  Clear histogram
    memset(pHistogram, 0, (m_iLevels * sizeof(long)));

//  Create regions of interest
    CreateROI(
        0, 0,
        m_iTileWidth, m_iTileHeight);
```

```
    pImage->CreateROI(
        0, 0,
        m_iTileWidth, m_iTileHeight);

//  Rows
    for(yt = 0; yt < iYTiles; yt++)
    //  Cols
        for(xt = 0; xt < iXTiles; xt++)
        {
        //  Set regions of interest
            SetROI(
                (xt * m_iTileWidth), (yt * m_iTileHeight),
                m_iTileWidth, m_iTileHeight);
            pImage->SetROI(
                (xt * m_iTileWidth), (yt * m_iTileHeight),
                m_iTileWidth, m_iTileHeight);

        //  Tile from image
            Tile.Copy(this);

        //  Rows
            for(yp = 0; yp < m_iTileHeight; yp++)
            //  Cols
                for(xp = 0; xp < m_iTileWidth; xp++)
                {
                    Tile.GetPixel(xp, yp, &byPixel);
                    byPixel = (BYTE)T[byPixel];
                    Tile.PutPixel(xp, yp, &byPixel);

                //  Histogram
                    pHistogram[byPixel]++;
                }

        //  Save tile
            pImage->Copy(&Tile);
        }

//  Delete regions of interest
    DeleteROI();
    pImage->DeleteROI();

//  Delete F function
    delete [] F;

//  Delete T function
    delete [] T;
}

////////////////////////////////////////////////////////////////////////
//  Dyadic arithmetic operations

/*
////////////////////////////////////////////////////////////////////////
// Name:        Subtract_Abs
// Purpose:     Subtracts pixel values of one image from those of another
                image and computes absolute pixel values of resultant image
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
        CImage* pImageC - Second source image
        CImage* pImageX - Resultant image
```

```
// Notes:
*/
void CImage::Subtract_Abs(CImage* pImageC, CImage* pImageX)
{
//  Number of tiles
    int iXTiles = (GetWidth() / m_iTileWidth);  //  Horizontal
    int iYTiles = (GetHeight() / m_iTileHeight);    //  Vertical

//  DN
    BYTE byPixel;
    BYTE byPixelC;

//  Counters
    register int xt;
    register int yt;
    register int xp;
    register int yp;

//  Tiles
    CMImage Tile(m_iTileWidth, m_iTileHeight);  //  this
    CMImage TileC(m_iTileWidth, m_iTileHeight);

//  Create regions of interest
    CreateROI(
        0, 0,
        m_iTileWidth, m_iTileHeight);
    pImageC->CreateROI(
        0, 0,
        m_iTileWidth, m_iTileHeight);
    pImageX->CreateROI(
        0, 0,
        m_iTileWidth, m_iTileHeight);

//  Rows
    for(yt = 0; yt < iYTiles; yt++)
    //  Cols
        for(xt = 0; xt < iXTiles; xt++)
        {
        //  Set regions of interest
            SetROI(
                (xt * m_iTileWidth), (yt * m_iTileHeight),
                m_iTileWidth, m_iTileHeight);
            pImageC->SetROI(
                (xt * m_iTileWidth), (yt * m_iTileHeight),
                m_iTileWidth, m_iTileHeight);
            pImageX->SetROI(
                (xt * m_iTileWidth), (yt * m_iTileHeight),
                m_iTileWidth, m_iTileHeight);

        //  Tile from reference image
            Tile.Copy(this);

        //  Tile from image to correct
            TileC.Copy(pImageC);

        //  Rows
            for(yp = 0; yp < m_iTileHeight; yp++)
            //  Cols
                for(xp = 0; xp < m_iTileWidth; xp++)
                {
                    Tile.GetPixel(xp, yp, &byPixel);
                    TileC.GetPixel(xp, yp, &byPixelC);
```

```
                byPixel = abs(byPixel - byPixelC);
                Tile.PutPixel(xp, yp, &byPixel);
            }

        //  Set tile
            pImageX->Copy(&Tile);
        }

//  Delete regions of interest
    DeleteROI();
    pImageC->DeleteROI();
    pImageX->DeleteROI();
}

/*
//////////////////////////////////////////////////////////////////////
// Name:        Subtract_AddScalar
// Purpose:     Subtracts pixel values of one image from those of another
                image and adds a constant to pixel values of the resultant image
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
        CImage* pImageC - Second source image
        BYTE byScalar - Constant
        CImage* pImageX - Resultant image
// Notes:
*/
void CImage::Subtract_AddScalar(CImage* pImageC, BYTE byScalar,
        CImage* pImageX)
{
//  Number of tiles
    int iXTiles = (GetWidth() / m_iTileWidth);  //  Horizontal
    int iYTiles = (GetHeight() / m_iTileHeight);    //  Vertical

//  DN
    BYTE byPixel;
    BYTE byPixelC;

//  Counters
    register int xt;
    register int yt;
    register int xp;
    register int yp;

//  Tiles
    CMImage Tile(m_iTileWidth, m_iTileHeight);  //  this
    CMImage TileC(m_iTileWidth, m_iTileHeight);

//  Create regions of interest
    CreateROI(
        0, 0,
        m_iTileWidth, m_iTileHeight);
    pImageC->CreateROI(
        0, 0,
        m_iTileWidth, m_iTileHeight);
    pImageX->CreateROI(
        0, 0,
        m_iTileWidth, m_iTileHeight);

//  Rows
    for(yt = 0; yt < iYTiles; yt++)
```

```
    //  Cols
        for(xt = 0; xt < iXTiles; xt++)
        {
        //  Set regions of interest
            SetROI(
                (xt * m_iTileWidth), (yt * m_iTileHeight),
                m_iTileWidth, m_iTileHeight);
            pImageC->SetROI(
                (xt * m_iTileWidth), (yt * m_iTileHeight),
                m_iTileWidth, m_iTileHeight);
            pImageX->SetROI(
                (xt * m_iTileWidth), (yt * m_iTileHeight),
                m_iTileWidth, m_iTileHeight);

        //  Tile from reference image
            Tile.Copy(this);

        //  Tile from image to correct
            TileC.Copy(pImageC);

        //  Rows
            for(yp = 0; yp < m_iTileHeight; yp++)
            //  Cols
                for(xp = 0; xp < m_iTileWidth; xp++)
                {
                    Tile.GetPixel(xp, yp, &byPixel);
                    TileC.GetPixel(xp, yp, &byPixelC);
                    byPixel = esc(
                        0,  //  DN = 0 (Min.)
                        (byPixel - byPixelC + byScalar),
                        (m_iLevels - 1));   //  DN = 255 (Max.)
                    Tile.PutPixel(xp, yp, &byPixel);
                }

        //  Set tile
            pImageX->Copy(&Tile);
        }

//  Delete regions of interest
    DeleteROI();
    pImageC->DeleteROI();
    pImageX->DeleteROI();
}
```

DImage.h

```
// DImage.h: interface for the CDImage class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_DIMAGE_H_INCLUDED_)
#define AFX_DIMAGE_H_INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

//  Classes
class CImage;
class CMImage;
```

```
class CDImage : public CImage
{
public:
    CDImage(int iWidth, int iHeight);
    CDImage(const CDImage& Image);
    CDImage(const CMImage& Image);
    virtual ~CDImage();

//  User functions
//
    virtual int LoadRAWFile(const char* pFileName);
    virtual void SaveRAWFile(const char* pFileName);

//  For Tcl/Tk interface
    virtual void SavePPMFile(const char* pFileName);

protected:

//  Virtual memory system
    static long m_lFiles;

//  Tiling system
    CImageInfo* m_pImageInfo;
};

#endif // !defined(AFX_DIMAGE_H_INCLUDED_)
```

DImage.cpp

```
// DImage.cpp: implementation of the CDImage class.
//
//////////////////////////////////////////////////////////////////////

#include "IPSoft.h"

#include "ImageInfo.h"  //  Image information (real information)

long CDImage::m_lFiles = -1;

/*
//////////////////////////////////////////////////////////////////////
// Name:        GetTileToRead
// Purpose:     Returns tile in the tile info structure of shape image.
// Context:     i42aresv Soft
// Returns:     IPLStatus - IPL_StsOk: Success
// Parameters:
        IplImage* pImage - Image to be connected with tile data
        int iXIndex, int iYIndex - Address of tile
// Notes:
        The tile is a copy from a source image storage.
        Image storage is pImage->imageId field.
*/
IPLStatus GetTileToRead(IplImage* pImage, int iXIndex, int iYIndex)
{
//  Image information (real information)
    CImageInfo* pImageInfo = (CImageInfo *)pImage->imageId;

//  File
    ifstream* pFile = (ifstream *)pImageInfo->GetFile();
//  Image size
    int iImageWidth  = pImageInfo->GetWidth();
```

```
    int iImageHeight = pImageInfo->GetHeight();

//  Tile size
    int iTileWidth  = (iImageWidth  - (iXIndex * pImage->tileInfo->width));
    iTileWidth  = (iTileWidth  < pImage->tileInfo->width)  ?
        iTileWidth  : pImage->tileInfo->width;
    int iTileHeight = (iImageHeight - (iYIndex * pImage->tileInfo->height));
    iTileHeight = (iTileHeight < pImage->tileInfo->height) ?
        iTileHeight : pImage->tileInfo->height;

//  Create tile
    IplImage* pTile = iplCreateImageHeader(
        1,
        0,
        IPL_DEPTH_8U,
        "GRAY",
        "GRAY",
        IPL_DATA_ORDER_PIXEL,
        IPL_ORIGIN_TL,
        IPL_ALIGN_DWORD,
        pImage->tileInfo->width,
        pImage->tileInfo->height,
        NULL,
        NULL,
        NULL,
        NULL);
    iplAllocateImage(pTile, 1, 0);

//  Offset (bytes)
    //  streampos <=> long
    streampos lPosition = (iXIndex * pImage->tileInfo->width) +
        (iYIndex * pImage->tileInfo->height * iImageWidth);

//  Counter
    register int i;

//  Rows
    for(i = 0; i < iTileHeight; i++)
    {
        pFile->seekg(lPosition);
        pFile->read((pTile->imageData + i * pImage->tileInfo->width), iTileWidth);

        lPosition += iImageWidth;   //  Next row
    }
    pImage->tileInfo->tileData = pTile->imageData;
    iplDeallocate(pTile, IPL_IMAGE_HEADER); //  Deallocate image header

    return IPL_StsOk;
}

/*
//////////////////////////////////////////////////////////////////////
// Name:        GetTileToWrite
// Purpose:     Returns empty tile in the tile info structure of shape image.
// Context:     i42aresv Soft
// Returns:     IPLStatus - IPL_StsOk if success
// Parameters:
        IplImage* pImage - Image to be connected with tile data
// Notes:
*/
IPLStatus GetTileToWrite(IplImage* pImage)
{
```

```
//  Create tile
    IplImage* pTile = iplCreateImageHeader(
        1,
        0,
        IPL_DEPTH_8U,
        "GRAY",
        "GRAY",
        IPL_DATA_ORDER_PIXEL,
        IPL_ORIGIN_TL,
        IPL_ALIGN_DWORD,
        pImage->tileInfo->width,
        pImage->tileInfo->height,
        NULL,
        NULL,
        NULL,
        NULL);
    iplAllocateImage(pTile, 1, 0);
    pImage->tileInfo->tileData = pTile->imageData;
    iplDeallocate(pTile, IPL_IMAGE_HEADER); //  Deallocate image header

    return IPL_StsOk;
}

/*
//////////////////////////////////////////////////////////////////////
// Name:        ReleaseTile
// Purpose:     Put tile into a destination image file
// Context:     i42aresv Soft
// Returns:     IPLStatus - IPL_StsOk: Success
// Parameters:
        IplImage* pImage - Image connected with tile data
        int iXIndex, int iYIndex - Address of tile
// Notes:
        The tile is the tile info structure of the shape image.
        Tile is deleted after an operation. The tileData field
        is set to NULL.
        Image storage is pImage->imageId field.
*/
IPLStatus ReleaseTile(const IplImage* pImage, int iXIndex, int iYIndex)
{
//  Image information (real information)
    CImageInfo* pImageInfo = (CImageInfo *)pImage->imageId;

//  File
    ofstream* pFile = (ofstream *)pImageInfo->GetFile();
//  Image size
    int iImageWidth  = pImageInfo->GetWidth();
    int iImageHeight = pImageInfo->GetHeight();

//  Tile size
    int iTileWidth  = (iImageWidth  - (iXIndex * pImage->tileInfo->width));
    iTileWidth  = (iTileWidth  < pImage->tileInfo->width)  ?
        iTileWidth  : pImage->tileInfo->width;
    int iTileHeight = (iImageHeight - (iYIndex * pImage->tileInfo->height));
    iTileHeight = (iTileHeight < pImage->tileInfo->height) ?
        iTileHeight : pImage->tileInfo->height;

//  Create tile
    IplImage* pTile = iplCreateImageHeader(
        1,
        0,
        IPL_DEPTH_8U,
```

```
        "GRAY",
        "GRAY",
        IPL_DATA_ORDER_PIXEL,
        IPL_ORIGIN_TL,
        IPL_ALIGN_DWORD,
        pImage->tileInfo->width,
        pImage->tileInfo->height,
        NULL,
        NULL,
        NULL,
        NULL);
    pTile->imageData = pImage->tileInfo->tileData;

//  Offset (bytes)
    //  streampos <=> long
    streampos lPosition = (iXIndex * pImage->tileInfo->width) +
        (iYIndex * pImage->tileInfo->height * iImageWidth);

//  Counter
    register int i;

//  Rows
    for(i = 0; i < iTileHeight; i++)
    {
        pFile->seekp(lPosition);
        pFile->write((pTile->imageData + i * pImage->tileInfo->width), iTileWidth);

        lPosition += iImageWidth;   //  Next row
    }
    pImage->tileInfo->tileData = NULL;
    iplDeallocateImage(pTile);  //  Deallocate image data
    iplDeallocate(pTile, IPL_IMAGE_HEADER); //  Deallocate image header

    return IPL_StsOk;
}

/*
//////////////////////////////////////////////////////////////////////
// Name:        CallBack
// Purpose:     CallBack function for this tiling system
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
        IplImage* pImage - Image connected with tile data
        int iXIndex, int iYIndex - Address of tile
        int iMode - Tile access mode
//  Notes:
*/
void __stdcall CallBack(const IplImage* pImage, int iXIndex, int iYIndex, int iMode)
{
//  Check image before calling
    if((pImage == NULL) || (pImage->tileInfo == NULL) || (pImage->imageId == NULL))
        return;

//  Operation
    switch(iMode)
    {
    case IPL_GET_TILE_TO_READ:
        GetTileToRead((IplImage *)pImage, iXIndex, iYIndex);
        break;
    case IPL_GET_TILE_TO_WRITE:
        GetTileToWrite((IplImage *)pImage);
```

```
        break;
    case IPL_RELEASE_TILE:
        ReleaseTile(pImage, iXIndex, iYIndex);
        break;
    }
}

////////////////////////////////////////////////////////////////////
// Construction/Destruction

/*
////////////////////////////////////////////////////////////////////
// Name:        CDImage
// Purpose:     Constructor
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
        int iWidth - Image width
        int iHeight - Image height
// Notes:
*/
CDImage::CDImage(int iWidth, int iHeight)
    : CImage(iWidth, iHeight)
{
//  New temporal file
    m_lFiles++;

//  File name
    char* pFileName = new char[256];
    sprintf(pFileName, "_%10d_.RAW", m_lFiles);

//  Image information (real information)
    m_pImageInfo = new CImageInfo(iWidth, iHeight, pFileName);

//  Create tile information
    m_pImage->imageId = (void *)m_pImageInfo;
    m_pImage->tileInfo = iplCreateTileInfo(
        CallBack,
        (void *)pFileName,
        m_iTileWidth,
        m_iTileHeight);
}

/*
////////////////////////////////////////////////////////////////////
// Name:        CDImage
// Purpose:     Copy constructor
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
        const CDImage& Image - Source image
// Notes:
*/
CDImage::CDImage(const CDImage& Image)
    : CImage(Image.m_pImage->width, Image.m_pImage->height)
{
//  New temporal file
    m_lFiles++;

//  File name
    char* pFileName = new char[256];
    sprintf(pFileName, "_%10d_.RAW", m_lFiles);
```

```
//  Image information (real information)
    m_pImageInfo = new CImageInfo(
        Image.m_pImage->width,
        Image.m_pImage->height,
        pFileName);

//  Create tile information
    m_pImage->imageId = (void *)m_pImageInfo;
    m_pImage->tileInfo = iplCreateTileInfo(
        CallBack,
        (void *)pFileName,
        m_iTileWidth,
        m_iTileHeight);

//  Copy data
    Copy((CImage *)&Image);
}

/*
//////////////////////////////////////////////////////////////////////
// Name:        CDImage
// Purpose:     Copy constructor
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
        const CMImage& Image - Source image
// Notes:
        Copy memory image to disk image
*/
CDImage::CDImage(const CMImage& Image)
    : CImage(((CMImage *)&Image)->GetWidth(), ((CMImage *)&Image)->GetHeight())
{
//  New temporal file
    m_lFiles++;

//  File name
    char* pFileName = new char[256];
    sprintf(pFileName, "_%10d_.RAW", m_lFiles);

//  Image information (real information)
    m_pImageInfo = new CImageInfo(
        ((CDImage *)&Image)->m_pImage->width,
        ((CDImage *)&Image)->m_pImage->height,
        pFileName);

//  Create tile information
    m_pImage->imageId = (void *)m_pImageInfo;
    m_pImage->tileInfo = iplCreateTileInfo(
        CallBack,
        (void *)pFileName,
        m_iTileWidth,
        m_iTileHeight);

//  Copy data
    Copy((CImage *)&Image);
}

/*
//////////////////////////////////////////////////////////////////////
// Name:        ~CDImage
// Purpose:     Destructor
```

```
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
// Notes:
*/
CDImage::~CDImage()
{
//  Delete image information (real information)
    delete m_pImageInfo;

//  Remove temporal file and delete file name
    remove((char *)m_pImage->tileInfo->id);
    delete [] m_pImage->tileInfo->id;

//  Delete tile information
    iplDeleteTileInfo(m_pImage->tileInfo);
}

////////////////////////////////////////////////////////////////////
// Virtual functions

/*
////////////////////////////////////////////////////////////////////
// Name:        LoadRAWFile
// Purpose:     Loads a RAW file into disk image
// Context:     i42aresv Soft
// Returns:     int - 0: Success / -1: Failure
// Parameters:
        const char* pFileName - RAW file name
// Notes:
*/
int CDImage::LoadRAWFile(const char* pFileName)
{
    if(_access(pFileName, 0) == -1)
        return -1;

//  Data file
    fstream* pFile = new fstream(pFileName,
            ios::in | ios::out | ios::binary);

//  Buffer (row)
    LPBYTE pBuffer = new BYTE[m_pImage->width];

//  File offset
    streamoff lPosition = 0;

//  Rows
    register int i;
    for(i = 0; i < m_pImage->height; i++)
    {
        pFile->seekg(lPosition);
        m_pImageInfo->GetFile()->seekp(lPosition);

        pFile->read(pBuffer, m_pImage->width);
        m_pImageInfo->GetFile()->write(pBuffer, m_pImage->width);

        lPosition += m_pImage->width;   //  Next row
    }

//  Delete buffer
    delete [] pBuffer;
```

```
//  Close and delete file image
    pFile->close();
    delete pFile;

    return 0;
}

/*
//////////////////////////////////////////////////////////////////////
// Name:        SaveRAWFile
// Purpose:     Saves a disk image into RAW file
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
        const char* pFileName - RAW file name
// Notes:
*/
void CDImage::SaveRAWFile(const char* pFileName)
{
//  Data file
    fstream* pFile = new fstream(pFileName,
            ios::in | ios::out | ios::binary | ios::trunc);

//  Buffer (row)
    LPBYTE pBuffer = new BYTE[m_pImage->width];

//  File offset
    streamoff lPosition = 0;

//  Rows
    register int i;
    for(i = 0; i < m_pImage->height; i++)
    {
        m_pImageInfo->GetFile()->seekg(lPosition);
        pFile->seekp(lPosition);

        m_pImageInfo->GetFile()->read(pBuffer, m_pImage->width);
        pFile->write(pBuffer, m_pImage->width);

        lPosition += m_pImage->width;   //  Next row
    }

//  Delete buffer
    delete [] pBuffer;

//  Close and delete file image
    pFile->close();
    delete pFile;
}

/*
//////////////////////////////////////////////////////////////////////
// Name:        SavePPMFile
// Purpose:     Saves a disk image into PPM file
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
        const char* pFileName - PPM file name (RGB image)
// Notes:
*/
void CDImage::SavePPMFile(const char* pFileName)
{
```

```
//  Data file
    fstream* pFile = new fstream(pFileName,
            ios::out | ios::binary | ios::trunc);

//  Buffer
    char* pHeader = new char[128];

//  Write header
//

//  Magic number
    sprintf(pHeader, "P6\n");
    pFile->write(pHeader, strlen(pHeader));
//  Width/Height
    sprintf(pHeader, "%d %d\n", m_pImage->width, m_pImage->height);
    pFile->write(pHeader, strlen(pHeader));
//  Maximun value
    sprintf(pHeader, "255\n");
    pFile->write(pHeader, strlen(pHeader));

//  Delete buffer
    delete [] pHeader;

//  Buffer (row)
    LPBYTE pBuffer = new BYTE[m_pImage->width];

//  File offset
    streamoff lPosition = 0;

//  Counters
    register int i;
    register int j;

//  Rows
    for(j = 0; j < m_pImage->height; j++)
    {
        m_pImageInfo->GetFile()->seekg(lPosition);
        m_pImageInfo->GetFile()->read(pBuffer, m_pImage->width);
        lPosition += m_pImage->width;   //  Next row

    //  Cols
        for(i = 0; i < m_pImage->width; i++)
        {
        //  RED band
            pFile->put(*(pBuffer + i));
        //  GREEN band
            pFile->put(*(pBuffer + i));
        //  BLUE band
            pFile->put(*(pBuffer + i));
        }
    }

//  Delete buffer
    delete [] pBuffer;

//  Close and delete file image
    pFile->close();
    delete pFile;
}
```

MImage.h

```
// MImage.h: interface for the CMImage class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_MIMAGE_H_INCLUDED_)
#define AFX_MIMAGE_H_INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

//  Classes
class CImage;
class CDImage;

class CMImage : public CImage
{
public:
    CMImage(int iWidth, int iHeight);
    CMImage(const CMImage& Image);
    CMImage(const CDImage& Image);
    virtual ~CMImage();

//  Data exchange functions
//
    void PutPixel(int x, int y, BYTE* pPixel);
    void GetPixel(int x, int y, BYTE* pPixel);

//  User functions
//
    virtual int LoadRAWFile(const char* pFileName);
    virtual void SaveRAWFile(const char* pFileName);

//  For Tcl/Tk interface
    virtual void SavePPMFile(const char* pFileName);

    LPBYTE GetImageData();
};

#endif // !defined(AFX_MIMAGE_H_INCLUDED_)
```

MImage.cpp

```
// MImage.cpp: implementation of the CMImage class.
//
//////////////////////////////////////////////////////////////////////

#include "IPSoft.h"

//////////////////////////////////////////////////////////////////////
// Construction/Destruction

/*
//////////////////////////////////////////////////////////////////////
// Name:        CMImage
// Purpose:     Constructor
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
```

```
        int iWidth - Image width
        int iHeight - Image height
// Notes:
*/
CMImage::CMImage(int iWidth, int iHeight)
    : CImage(iWidth, iHeight)
{
//  Image data
    iplAllocateImage(m_pImage, 1, 0);
}

/*
/////////////////////////////////////////////////////////////////////
// Name:        CMImage
// Purpose:     Copy constructor
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
        const CMImage& Image - Source image
// Notes:
*/
CMImage::CMImage(const CMImage& Image)
    : CImage(Image.m_pImage->width, Image.m_pImage->height)
{
//  Image data
    iplAllocateImage(m_pImage, 1, 0);

//  Copy data
    Copy((CImage *)&Image);
}

/*
/////////////////////////////////////////////////////////////////////
// Name:        CMImage
// Purpose:     Copy constructor
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
        const CDImage& Image - Source image
// Notes:
        Copy disk image to memory image
*/
CMImage::CMImage(const CDImage& Image)
    : CImage(((CDImage *)&Image)->GetWidth(), ((CDImage *)&Image)->GetHeight())
{
//  Image data
    iplAllocateImage(m_pImage, 1, 0);

//  Copy data
    Copy((CImage *)&Image);
}

/*
/////////////////////////////////////////////////////////////////////
// Name:        ~CMImage
// Purpose:     Destructor
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
// Notes:
*/
CMImage::~CMImage()
```

```
{
    iplDeallocateImage(m_pImage);
}

////////////////////////////////////////////////////////////////////
// User functions

/*
////////////////////////////////////////////////////////////////////
// Name:        PutPixel
// Purpose:     Puts the value of pixel with coordinates (x,y)
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
        int x, int y - Pixel coordinates
        BYTE* pPixel - Pixel value
// Notes:
*/
void CMImage::PutPixel(int x, int y, BYTE* pPixel)
{
//  Are the pixel coordinates out of range ?
    if((x < 0) || (y < 0))
        return;
    if((x >= m_pImage->width) || (y >= m_pImage->height))
        return;
    iplPutPixel(m_pImage, x, y, pPixel);
}

/*
////////////////////////////////////////////////////////////////////
// Name:        GetPixel
// Purpose:     Gets the value of pixel with coordinates (x,y)
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
        int x, int y - Pixel coordinates
        BYTE* pPixel - Pixel value
// Notes:
*/
void CMImage::GetPixel(int x, int y, BYTE* pPixel)
{
//  Are the pixel coordinates out of range ?
    if((x < 0) || (y < 0))
        return;
    if((x >= m_pImage->width) || (y >= m_pImage->height))
        return;
    iplGetPixel(m_pImage, x, y, pPixel);
}

/*
////////////////////////////////////////////////////////////////////
// Name:        GetImageData
// Purpose:     Gets image data from IplImage struct
// Context:     i42aresv Soft
// Returns:     LPBYTE - Image data (RAW format)
// Parameters:
// Notes:
*/
LPBYTE CMImage::GetImageData()
{
    return (LPBYTE)m_pImage->imageData;
}
```

```
////////////////////////////////////////////////////////////////////
// Virtual functions

/*
////////////////////////////////////////////////////////////////////
// Name:        LoadRAWFile
// Purpose:     Loads a RAW file into memory image
// Context:     i42aresv Soft
// Returns:     int - 0: Success / -1: Failure
// Parameters:
        const char* pFileName - RAW file name
// Notes:
*/
int CMImage::LoadRAWFile(const char* pFileName)
{
    if(_access(pFileName, 0) == -1)
        return -1;

    fstream* pFile = new fstream(pFileName,
            ios::in | ios::out | ios::binary);

//  File offset
    streamoff lPosition = 0;

//  Rows
    register int i;
    for(i = 0; i < m_pImage->height; i++)
    {
        pFile->seekg(lPosition);
        pFile->read((m_pImage->imageData + i * m_pImage->width),
                m_pImage->width);

        lPosition += m_pImage->width;   //  Next row
    }

//  Close and delete file image
    pFile->close();
    delete pFile;

    return 0;
}

/*
////////////////////////////////////////////////////////////////////
// Name:        SaveRAWFile
// Purpose:     Saves a memory image into RAW file
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
        const char* pFileName - RAW file name
// Notes:
*/
void CMImage::SaveRAWFile(const char* pFileName)
{
    fstream* pFile = new fstream(pFileName,
            ios::in | ios::out | ios::binary | ios::trunc);

//  File offset
    streamoff lPosition = 0;

//  Rows
```

```
    register int i;
    for(i = 0; i < m_pImage->height; i++)
    {
        pFile->seekp(lPosition);
        pFile->write((m_pImage->imageData + i * m_pImage->width),
                m_pImage->width);

        lPosition += m_pImage->width;   //  Next row
    }

//  Close and delete file image
    pFile->close();
    delete pFile;
}

/*
//////////////////////////////////////////////////////////////////////
// Name:        SavePPMFile
// Purpose:     Saves a memory image into PPM file
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
        const char* pFileName - PPM file name (RGB image)
// Notes:
*/
void CMImage::SavePPMFile(const char* pFileName)
{
//  Data file
    fstream* pFile = new fstream(pFileName,
            ios::out | ios::binary | ios::trunc);

//  Buffer
    char* pHeader = new char[128];

//  Write header (17 bytes)
//

//  Magic number
    sprintf(pHeader, "P6\n");
    pFile->write(pHeader, strlen(pHeader));
//  Width/Height
    sprintf(pHeader, "%d %d\n", m_pImage->width, m_pImage->height);
    pFile->write(pHeader, strlen(pHeader));
//  Maximun value
    sprintf(pHeader, "255\n");
    pFile->write(pHeader, strlen(pHeader));

//  Delete buffer
    delete [] pHeader;

//  Counters
    register int i;
    register int j;

//  Rows
    for(j = 0; j < m_pImage->height; j++)
    {
    //  Cols
        for(i = 0; i < m_pImage->width; i++)
        {
        //  RED band
            pFile->put(*(m_pImage->imageData + j * m_pImage->width + i));
```

```
        //  GREEN band
            pFile->put(*(m_pImage->imageData + j * m_pImage->width + i));
        //  BLUE band
            pFile->put(*(m_pImage->imageData + j * m_pImage->width + i));
        }
    }

//  Close and delete file image
    pFile->close();
    delete pFile;
}
```

ImageInfo.h

```
// ImageInfo.h: interface for the CImageInfo class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_IMAGEINFO_H_INCLUDED_)
#define AFX_IMAGEINFO_H_INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include <fstream.h>

class CImageInfo
{
//  Image size (real size)
    int m_iWidth;
    int m_iHeight;

//  Image file
    fstream* m_pFile;

public:
    CImageInfo(int iWidth, int iHeight, const char* pFileName);
    virtual ~CImageInfo();

//  User functions
//
    int GetWidth();
    int GetHeight();
    fstream* GetFile();
};

#endif // !defined(AFX_IMAGEINFO_H_INCLUDED_)
```

ImageInfo.cpp

```
// ImageInfo.cpp: implementation of the CImageInfo class.
//
//////////////////////////////////////////////////////////////////////

#include "ImageInfo.h"

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
```

```
/*
////////////////////////////////////////////////////////////////////
// Name:        CImageInfo
// Purpose:     Constructor
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
        int iWidth - Image width
        int iHeight - Image height
        const char* pFileName - Image file name
// Notes:
*/
CImageInfo::CImageInfo(int iWidth, int iHeight, const char* pFileName)
{
//  Image size
    m_iWidth = iWidth;
    m_iHeight = iHeight;

//  Image file
    m_pFile = new fstream(pFileName, ios::in | ios::out | ios::binary);
}

/*
////////////////////////////////////////////////////////////////////
// Name:        ~CImageInfo
// Purpose:     Destructor
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
// Notes:
*/
CImageInfo::~CImageInfo()
{
//  Close and delete the image file
    m_pFile->close();
    delete m_pFile;
}

////////////////////////////////////////////////////////////////////
//  User functions

/*
////////////////////////////////////////////////////////////////////
// Name:        GetWidth
// Purpose:     Gets image width
// Context:     i42aresv Soft
// Returns:     int - Image width
// Parameters:
// Notes:
*/
CImageInfo::GetWidth()
{
    return m_iWidth;
}

/*
////////////////////////////////////////////////////////////////////
// Name:        GetHeight
// Purpose:     Gets image height
// Context:     i42aresv Soft
// Returns:     int - Image height
```

```
// Parameters:
// Notes:
*/
int CImageInfo::GetHeight()
{
    return m_iHeight;
}

/*
////////////////////////////////////////////////////////////////////
// Name:        GetFile
// Purpose:     Gets image file
// Context:     i42aresv Soft
// Returns:     fstream* - Image file
// Parameters:
// Notes:
*/
fstream* CImageInfo::GetFile()
{
    return m_pFile;
}
```

Polygon.h

```
// Polygon.h: interface for the CPolygon class.
//
////////////////////////////////////////////////////////////////////

#if !defined(AFX_POLYGON_H_)
#define AFX_POLYGON_H_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

//typedef struct {int x; int y; } POINT;

class CPolygon
{
//  Size
    int m_iSize;

//  Polygon
    POINT* m_pPolygon;

public:
    CPolygon(int Size, POINT* pPolygon);
    CPolygon(const CPolygon& Polygon);
    CPolygon(const CPolygon* Polygon);
    virtual ~CPolygon();

// User functions
//
    bool PointInPolygon(int x, int y);
    bool RectInPolygon(int x, int y, int iWidth, int iHeight);
};

#endif // !defined(AFX_POLYGON_H_)
```

Polygon.cpp

```
// Polygon.cpp: implementation of the CPolygon class.
//
//////////////////////////////////////////////////////////////////////

#include <IPSoft.h>

//////////////////////////////////////////////////////////////////////
// Construction/Destruction

/*
//////////////////////////////////////////////////////////////////////
// Name:        CPolygon
// Purpose:     Constructor
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
        int iSize - Size of vertex array
        POINT* pPolygon - Vertex array
// Notes:
*/
CPolygon::CPolygon(int iSize, POINT* pPolygon)
{
    m_iSize = iSize;
    m_pPolygon = new POINT[iSize];

//  Copy data
    memcpy(m_pPolygon, pPolygon, (sizeof(POINT) * iSize));
}

/*
//////////////////////////////////////////////////////////////////////
// Name:        CPolygon
// Purpose:     Copy constructor
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
        const CPolygon& Polygon - Source polygon
// Notes:
*/
CPolygon::CPolygon(const CPolygon& Polygon)
{
    m_iSize = Polygon.m_iSize;
    m_pPolygon = new POINT[Polygon.m_iSize];

//  Copy data
    memcpy(m_pPolygon, Polygon.m_pPolygon, (sizeof(POINT) * Polygon.m_iSize));
}

/*
//////////////////////////////////////////////////////////////////////
// Name:        CPolygon
// Purpose:     Copy constructor
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
        const CPolygon* Polygon - Source polygon
// Notes:
*/
CPolygon::CPolygon(const CPolygon* Polygon)
{
```

```
    m_iSize = Polygon->m_iSize;
    m_pPolygon = new POINT[Polygon->m_iSize];

//  Copy data
    memcpy(m_pPolygon, Polygon->m_pPolygon, (sizeof(POINT) * Polygon->m_iSize));
}

/*
//////////////////////////////////////////////////////////////////////
// Name:        ~CPolygon
// Purpose:     Destructor
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
// Notes:
*/
CPolygon::~CPolygon()
{
    delete [] m_pPolygon;
}

//////////////////////////////////////////////////////////////////////
// User functions

/*
//////////////////////////////////////////////////////////////////////
// Name:        PointInPolygon
// Purpose:     Checks whether the point specified is inside of the boundaries
                of the polygon.
// Context:     i42aresv Soft
// Returns:     bool - true: Inside / false: Outside
// Parameters:
        int x, int y - Coordinates of point
// Notes:
*/
bool CPolygon::PointInPolygon(int x, int y)
{
//  Controls
    int iLeftOfPoint = 0;
    int iRightOfPoint = 0;

//  X coordinate of the intersection
    double dXIntersection;

//  Closed polygon
    for(register int i = 1; i < m_iSize; i++)
    {
        if( ((m_pPolygon[(i - 1)].y <= y) && (m_pPolygon[i].y > y)) ||
            ((m_pPolygon[(i - 1)].y > y) && (m_pPolygon[i].y <= y)))
        {
            dXIntersection = m_pPolygon[(i - 1)].x +
                ((y - m_pPolygon[(i - 1)].y) *
                    (m_pPolygon[i].x - m_pPolygon[(i - 1)].x) /
                    (m_pPolygon[i].y - m_pPolygon[(i - 1)].y));
            if(dXIntersection < x)
                iLeftOfPoint++;
            if(dXIntersection > x)
                iRightOfPoint++;
        }
    }
    if((iLeftOfPoint % 2 == 1) && (iRightOfPoint % 2 == 1))
        return true;
```

```
    else
        return false;
}

/*
////////////////////////////////////////////////////////////////////
// Name:        RectInPolygon
// Purpose:     Determines whether any part of the rectangle specified is
                inside of the boundaries of the polygon.
// Context:     i42aresv Soft
// Returns:     bool - true: Inside / false: Outside
// Parameters:
        int x, int y - Offsets from the origin of the rectangular region
        int iWidth, int iHeight - Size of rectangular region
// Notes:
*/
bool CPolygon::RectInPolygon(int x, int y, int iWidth, int iHeight)
{
    if( (PointInPolygon(x, y) == true) ||
        (PointInPolygon(x, (y + iHeight - 1)) == true) ||
        (PointInPolygon((x + iWidth - 1), y) == true) ||
        (PointInPolygon((x + iWidth - 1), (y + iHeight - 1)) == true))
        return true;
    else
        return false;
}
```

Matrix.h

```
// Matrix.h: interface for the CMatrix class.
//
////////////////////////////////////////////////////////////////////

#if !defined(AFX_MATRIX_H_INCLUDED_)
#define AFX_MATRIX_H_INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include <iostream.h>
#include <stdlib.h>

//  IMPORTANT
//      T must be a signed type

template <class T> class CMatrix
{
    long m_lRows;
    long m_lCols;

    T*  m_Data; //  Matrix data

public:

////////////////////////////////////////////////////////////////////
// Construction/Destruction

    CMatrix(long lRows, long lCols)
    {
        if(lRows < 1)
```

```
        {
            cerr << "ERROR <CMatrix::CMatrix>: lRows < 1" << endl;
            exit(-1);
        }
        if(lCols < 1)
        {
            cerr << "ERROR <CMatrix::CMatrix>: lCols < 1" << endl;
            exit(-1);
        }

        m_lRows = lRows;
        m_lCols = lCols;

        long lSize = (m_lRows * m_lCols);

        m_Data = new T[lSize];
    }

    CMatrix(const CMatrix<T>& Matrix)
    {
        m_lRows = Matrix.m_lRows;
        m_lCols = Matrix.m_lCols;

        long lSize = (m_lRows * m_lCols);

        m_Data = new T[lSize];

    //  Copy data
        memcpy(m_Data, Matrix.m_Data, (sizeof(T) * lSize));
    };

    CMatrix(const CMatrix<T>* Matrix)
    {
        m_lRows = Matrix->m_lRows;
        m_lCols = Matrix->m_lCols;

        long lSize = (m_lRows * m_lCols);

        m_Data = new T[lSize];

    //  Copy data
        memcpy(m_Data, Matrix->m_Data, (sizeof(T) * lSize));
    };

    virtual ~CMatrix()
    {
        delete [] m_Data;
    };

////////////////////////////////////////////////////////////////////////
// Matrix operations

    long GetRows(void)
    {
        return m_lRows;
    };

    long GetCols(void)
    {
        return m_lCols;
    };
```

```
void RemoveRow(long lRow)
{
    if(m_lRows < 2)
    {
        cerr << "ERROR <CMatrix::RemoveRow>: (m_lRows < 1) || \
                (lRow > (m_lRows - 1)" << endl;
        exit(-1);
    }
    if((lRow < 0) || (lRow > (m_lRows - 1)))
    {
        cerr << "ERROR <CMatrix::RemoveRow>: (lRow < 0) || \
                (lRow > (m_lRows - 1))" << endl;
        exit(-1);
    }

    long lSize = ((m_lRows - 1) * m_lCols);

    T* Data = new T[lSize];

    register int i;
    register int j;
    register int k = 0;
    for(j = 0; j < m_lRows; j++)
    {
        if(j == lRow)
            continue;

        for(i = 0; i < m_lCols; i++)
        {
            *(Data + k) = *(m_Data + j * m_lCols + i);
            k++;
        }
    }
    delete [] m_Data;
    m_Data = Data;
    m_lRows--;
};

void RemoveCol(long lCol)
{
    if(m_lCols < 2)
    {
        cerr << "ERROR <CMatrix::RemoveCol>: m_lCols < 1" << endl;
        exit(-1);
    }
    if((lCol < 0) || (lCol > (m_lCols - 1)))
    {
        cerr << "ERROR <CMatrix::RemoveCol>: (lCols < 0) || \
                (lCol > (m_lCols - 1))" << endl;
        exit(-1);
    }

    long lSize = (m_lRows * (m_lCols - 1));

    T* Data = new T[lSize];

    register int i;
    register int j;
    register int k = 0;
    for(j = 0; j < m_lRows; j++)
        for(i = 0; i < m_lCols; i++)
        {
```

```
            if(i == lCol)
                continue;

            *(Data + k) = *(m_Data + j * m_lCols + i);
            k++;
        }
    delete [] m_Data;
    m_Data = Data;
    m_lCols--;
};

T Determinant()     //  Matrix determinant
{
    if(m_lRows != m_lCols)
    {
        cerr << "ERROR <CMatrix::Determinant>: m_lRows != m_lCols" << endl;
        exit(-1);
    }

    T Determinant;  //  Result
    T a0i;
    T A0i;

    if((m_lRows == 1) && (m_lCols == 1))
        Determinant = *m_Data;
    else
    {
        Determinant = (T)0;
        register int i;
        for(i = 0; i < m_lCols; i ++)
        {
            a0i = *(m_Data + 0 * m_lCols + i);  //  ai0 element
            A0i = pow(-1, i) * SubMatrix(0, i).Determinant();
            Determinant += a0i * A0i;
        }
    }
    return Determinant;
};

CMatrix<T> SubMatrix(long lRow, long lCol)  //  Matrix constructed from A by
                                            //  removing row i and column j
{
    CMatrix<T> SubMatrix(this);

    SubMatrix.RemoveRow(lRow);
    SubMatrix.RemoveCol(lCol);

    return SubMatrix;
};

////////////////////////////////////////////////////////////////////////////
// CMatrix operators

    CMatrix<T>& operator=(const CMatrix<T>& Matrix)
    {
        if((m_lRows != Matrix.m_lRows) || (m_lCols != Matrix.m_lCols))
        {
            cerr << "ERROR <CMatrix::operator=>: (m_lRows != Matrix.m_lRows) || \
                    (m_lCols != Matrix.m_lCols)" << endl;
            exit(-1);
        }
```

```
            long lSize = (m_lRows * m_lCols);
            memcpy(m_Data, Matrix.m_Data, (sizeof(T) * lSize)); //  Copy data

            return *this;
        };

        T& operator()(long lRow, long lCol)
        {
            if((lRow < 0) || (lRow > (m_lRows - 1)))
            {
                cerr << "ERROR <CMatrix::operator()>: (lRow < 0) || \
                        (lRow > (m_lRows - 1))" << endl;
                exit(-1);
            }
            if((lCol < 0) || (lCol > (m_lCols - 1)))
            {
                cerr << "ERROR <CMatrix::operator()>: (lCols < 0) || \
                        (lCol > (m_lCols - 1))" << endl;
                exit(-1);
            }

            return *(m_Data + lRow * m_lCols + lCol);
        };

    #ifdef _DEBUG
        void Print(void)    //  Print matrix data
        {
            cout << endl << "A(" << m_lRows << ',' << m_lCols << ")" << endl;

            register int i;
            register int j;
            for(j = 0; j < m_lRows; j ++)
            {
                for(i = 0; i < m_lCols; i ++)
                    cout << "a" << j << i << "=" << *(m_Data + j * m_lCols + i) << '\t';
                cout << endl;
            }
        };
    #endif

    };

    typedef CMatrix<long>   CLMatrix;
    typedef CMatrix<double> CDMatrix;

    #endif // !defined(AFX_MATRIX_H_INCLUDED_)
```

Matrix.cpp

```
// Matrix.cpp: implementation of the CMatrix class.
//
//////////////////////////////////////////////////////////////////////

#include "Matrix.h"

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
/*
CMatrix::CMatrix()
{
```

```
}

CMatrix::~CMatrix()
{

}
*/
```

FastFormat.h

```
// FastFormat.h: interface for the CFastFormat class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_FASTFORMAT_H_INCLUDED_)
#define AFX_FASTFORMAT_H_INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include <fstream.h>

class CFastFormat
{
// Private attributes and operations
    static enum
    {
        RecordSize  = 1536  //  Record size
    };

    static enum     //  Cols
    {
        StartByte = 0,
        Size = 1
    };

//////////////////////////////////////////////////////////////////////////
//  Administrative record

    static const int m_iARField[][2];   //  Field start byte, end byte and size
    static enum     //  Rows
    {
    //  83  Number of pixels per image line
        PixelsPerLine = 83,
    //  87  Number of lines in the output image
        NumberOfLines = 87,
    };
    static char* GetARField(fstream* fHeader, int iField);

//////////////////////////////////////////////////////////////////////////
//  Radiometric record

//  Fields
    static const int m_iRRField[][2];   //  Field start byte, end byte and size
    static enum     //  Rows
    {
    //  4   Bias for first band on this tape
        Bias1 = 4,
    //  6   Gain for first band on this tape
        Gain1 = 6,
```

```
    };
    static char* GetRRField(fstream* fHeader, int iField);

////////////////////////////////////////////////////////////////////////
//  Geometric record

//  Fields
    static const int m_iGRField[][2];   //  Field start byte, end byte and size
    static enum     //  Rows
    {
    //  53  Easting of upper left corner of image in projections units
        ULEasting = 53,
    //  55  Northing of upper left corner of image in projections units
        ULNorthing = 55,
    //  64  Easting of upper right corner of image in projections units
        UREasting = 64,
    //  66  Northing of upper right corner of image in projections units
        URNorthing = 66,
    //  75  Easting of lower right corner of image in projections units
        LREasting = 75,
    //  77  Northing of lower right corner of image in projections units
        LRNorthing = 77,
    //  86  Easting of lower left corner of image in projections units
        LLEasting = 86,
    //  88  Northing of lower left corner of image in projections units
        LLNorthing = 88
    };
    static char* GetGRField(fstream* fHeader, int iField);

    ifstream* m_pFile;

public:
    CFastFormat(const char* pFileName);
    virtual ~CFastFormat();

//  Add user functions to get file fields

};

#endif // !defined(AFX_FASTFORMAT_H_INCLUDED_)
```

FastFormat.cpp

```
// FastFormat.cpp: implementation of the CFastFormat class.
//
//////////////////////////////////////////////////////////////////////

#include "FastFormat.h"

////////////////////////////////////////////////////////////////////////
// Administrative record fields

const int CFastFormat::m_iARField[][2] =
{
//  To access by field name
    {0, 0},
//  1   "PRODUCTbIDb="
//  2   Product order number in yydddnnn-cc format
    {0, 0}, {13, 11},   // (A11)
//  3   "bLOCATIONb="
//  4   First scene location path/row/fraction/subscene in ppp/rrrffss format
```

```
//      where ppp = Path(096), rrr = Row(051), ff=ShiftPer(20) and ss = Subscene
//      or Quadrant Number (e.g. 02)
    {0, 0}, {35, 17},   // (A17)
//  5   "bACQUISITIONbDATEb="
//  6   First scene acquisition date in yyyyddmm format
    {0, 0}, {71, 8},    // (A8)
//  7   Blank fill
//  8   Carriage return
//  9   "SATELLITEb="
//  10  First scene satellite name: L4, L5, 1B, 1C
    {0, 0}, {0, 0}, {0, 0}, {92, 10},   // (A10)
//  11  "bSENSORb="
//  12  First scene sensor name: TM, LISS1, LISS2, LISS3, PAN, WIFS
    {0, 0}, {111, 10},  // (A10)
//  13  "bSENSORbMODEb="
//  14  First scene sensor mode
    {0, 0}, {135, 6},   // (A6)
//  15  "bLOOKbANGLEb="
//  16  First scene off-nadir angle in degrees
    {0, 0}, {154, 6},   // (F6.2)
//  17  Carriage return
//  18  Blank fill
//  19  "bLOCATIONb="
//  20  Second scene location path/row/fraction/subscene in ppp/rrrffss format
//      where ppp = Path(096), rrr = Row(051), ff=ShiftPer(20) and ss = Subscene
//      or Quadrant Number (e.g. 02)
    {0, 0}, {0, 0}, {0, 0}, {195, 17},  // (A17)
//  21  "bACQUISITIONbDATEb="
//  22  Second scene acquisition date in yyyyddmm format
    {0, 0}, {231, 8},   // (A8)
//  23  Blank fill
//  24  Carriage return
//  25  "SATELLITEb="
//  26  Second scene satellite name: L4, L5, 1B, 1C
    {0, 0}, {0, 0}, {0, 0}, {252, 10},  // (A10)
//  27  "bSENSORb="
//  28  Second scene sensor name: TM, LISS1, LISS2, LISS3, PAN, WIFS
    {0, 0}, {271, 10},  // (A10)
//  29  "bSENSORbMODEb="
//  30  Second scene sensor mode
    {0, 0}, {295, 6},   // (A6)
//  31  "bLOOKbANGLEb="
//  32  Second scene off-nadir angle in degrees
    {0, 0}, {314, 6},   // (F6.2)
//  33  Carriage return
//  34  Blank fill
//  35  "bLOCATIONb="
//  36  Third scene location path/row/fraction/subscene in ppp/rrrffss format
//      where ppp = Path(096), rrr = Row(051), ff=ShiftPer(20) and ss = Subscene
//      or Quadrant Number (e.g. 02)
    {0, 0}, {0, 0}, {0, 0}, {355, 17},  // (A17)
//  37  "bACQUISITIONbDATEb="
//  38  Third scene acquisition date in yyyyddmm format
    {0, 0}, {391, 8},   // (A8)
//  39  Blank fill
//  40  Carriage return
//  41  "SATELLITEb="
//  42  Third scene satellite name: L4, L5, 1B, 1C
    {0, 0}, {0, 0}, {0, 0}, {412, 10},  // (A10)
//  43  "bSENSORb="
//  44  Third scene sensor name: TM, LISS1, LISS2, LISS3, PAN, WIFS
    {0, 0}, {431, 10},  // (A10)
```

```
//  45  "bSENSORbMODEb="
//  46  Third scene sensor mode
    {0, 0}, {455, 6},   // (A6)
//  47  "bLOOKbANGLEb="
//  48  Third scene off-nadir angle in degrees
    {0, 0}, {474, 6},   // (F6.2)
//  49  Carriage return
//  50  Blank fill
//  51  "bLOCATIONb="
//  52  Fourth scene location path/row/fraction/subscene in ppp/rrrffss format
//      where ppp = Path(096), rrr = Row(051), ff=ShiftPer(20) and ss = Subscene
//      or Quadrant Number (e.g. 02)
    {0, 0}, {0, 0}, {0, 0}, {515, 17},  // (A17)
//  53  "bACQUISITIONbDATEb="
//  54  Fourth scene acquisition date in yyyyddmm format
    {0, 0}, {551, 8},   // (A8)
//  55  Blank fill
//  56  Carriage return
//  57  "SATELLITEb="
//  58  Fourth scene satellite name: L4, L5, 1B, 1C
    {0, 0}, {0, 0}, {0, 0}, {572, 10},  // (A10)
//  59  "bSENSORb="
//  60  Fourth scene sensor name: TM, LISS1, LISS2, LISS3, PAN, WIFS
    {0, 0}, {591, 10},  // (A10)
//  61  "bSENSORbMODEb="
//  62  Fourth scene sensor mode
    {0, 0}, {615, 6},   // (A6)
//  63  "bLOOKbANGLEb="
//  64  Third scene off-nadir angle in degrees
    {0, 0}, {634, 6},   // (F6.2)
//  65  Carriage return
//  66  "PRODUCTbTYPEb="
//  67  Product type: 'MAPbORIENT0dbbbbbb', 'ORBITbORIENTEDbbbb'
    {0, 0}, {0, 0}, {655, 18},  // (A18)
//  68  "bPRODUCTbSIZEb="
//  69  Product size: 'FULLbSCENE', 'SUBSCENEbb', 'MAPbSHEETb', 'QUADRANT'
    {0, 0}, {688, 10},  // (A10)
//  70  Blank fill
//  71  Carriage return
//  72  "TYPEbOFbPROCESSINGb="
//  73  Type of processing used: 'SYSTEMAT-ICb', 'PRECISIONbb', 'TERRAINbbbb',
//      'RADIOMETRIC", 'RAWbbbbbbbbb'
    {0, 0}, {0, 0}, {0, 0}, {741, 11},  // (A11)
//  74  "bRESAMPLINGb="
//  75  Resampling algorithm used: 'CC', 'NN'
    {0, 0}, {765, 2},   // (A2)
//  76  Blank fill
//  77  Carriage return
//  78  "VOLUMEb#/#bINbSETb="
//  79  Tape volume number in tape set (for multi-volume image)
    {0, 0}, {0, 0}, {0, 0}, {820, 2},   // (I2)
//  80  "/"
//  81  Number of volumes in tape set (for multi-volume image)
    {0, 0}, {823, 2},   // (I2)
//  82  "bPIXELSbPERbLINEb="
//  83  Number of pixels per image line
    {0, 0}, {843, 5},   // (I5)
//  84  "bLINESbPERbBANDb="
//  85  Number of lines on this volume
    {0, 0}, {865, 5},   // (I5)
//  86  "/"
//  87  Number of lines in the output image
```

```
    {0, 0}, {871, 5},   // (I5)
//  88  Blank fill
//  89  Carriage return
//  90  "STARTbLINEb#b="
//  91  First image line number on this volume (for multi-volume image)
    {0, 0}, {0, 0}, {0, 0}, {895, 5},   // (I5)
//  92  "bBLOCKINGbFACTORb="
//  93  Tape blocking factor
    {0, 0}, {918, 2},   // (I2)
//  94  "bRECORDbLENGTHb="
//  95  Length of physical file record in bytes
    {0, 0}, {936, 5},   // (I5)
//  96  "bPIXELbSIZEb="
//  97  Pixel size in meters
    {0, 0}, {954, 6},   // (F6.2)
//  98  Carriage return
//  99  "OUTPUTbBITSbPERbPIXELb="
//  100 Output bits per pixel
    {0, 0}, {0, 0}, {984, 2},   // (I2)
//  101 "bACQUIREDbBITSbPERbPIXELb="
//  102 Acquired bits per pixel
    {0, 0}, {1012, 2},  // (I2)
//  103 Blank fill
//  104 Carriage return
//  105 "BANDSbPRESENTb="
//  106 Image bands present on this volume
    {0, 0}, {0, 0}, {0, 0}, {1056, 32}, // (A32)
//  107 "PRODUCTbCODEb="
//  108 Product code
    {0, 0}, {1103, 9},  // (A9)
//  109 Blank fill
//  110 Carrige return
//  111 "VERSIONbNOb="
//  112 Software version
    {0, 0}, {0, 0}, {0, 0}, {1133, 12}, // (A12)
//  113 Blank fill
//  114 "ACQUISITIONbTIMEb="
//  115 Time in HH:MM:SS:mmm
    {0, 0}, {0, 0}, {1171, 12}, // (A12)
//  116 Blank fill
//  117 Carrige return
//  118 "GENERATINGbCOUNTRYb="
//  119 Generating country name
    {0, 0}, {0, 0}, {0, 0}, {1221, 12}, // (A12)
//  120 Blank fill
//  121 "GENERATINGbAGENCYb="
//  122 Generating agency name
    {0, 0}, {0, 0}, {1255, 8},  // (A8)
//  123 Blank fill
//  124 Carriage return
//  125 "GENERATINGbFACILITYb="
//  126 Facility name
    {0, 0}, {0, 0}, {0, 0}, {1302, 5},  // (A5)
//  127 Blank fill
//  128 Carriage return
//  129 Blank fill
//  130 Carriage return
//  131 Blank fill
//  132 Carriage return
//  133 "REVbbbbbbbbbbbb"
//  134 Format version code (A-Z). This document describes version
    {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {1536, 1}   // (A1)
```

```
};

////////////////////////////////////////////////////////////////////////////
// Radiometric record fields

const int CFastFormat::m_iRRField[][2] =
{
//  To access by field name
    {0, 0},
//  1   "BIASESbANDbGAINSbINbTHEbBANDbORDERbASbONbTHISbTAPE"
//  2   Blank fill
//  3   Carriage return
//  4   Bias for first band on this tape
    {0, 0}, {0, 0}, {0, 0}, {81, 24},   // (D24.15)
//  5   Blank fill
//  6   Gain for first band on this tape
    {0, 0}, {106, 24},  // (D24.15)
//  7   Blank fill
//  8   Carriage return
//  9   Bias for second band on this tape
    {0, 0}, {0, 0}, {161, 24},  // (D24.15)
//  10  Blank fill
//  11  Gain for second band on this tape
    {0, 0}, {186, 24},  // (D24.15)
//  12  Blank fill
//  13  Carriage return
//  14  Bias for third band on this tape
    {0, 0}, {0, 0}, {241, 24},  // (D24.15)
//  15  Blank fill
//  16  Gain for third band on this tape
    {0, 0}, {266, 24},  // (D24.15)
//  17  Blank fill
//  18  Carriage return
//  19  Bias for fourth band on this tape
    {0, 0}, {0, 0}, {321, 24},  // (D24.15)
//  20  Blank fill
//  21  Gain for fourth band on this tape
    {0, 0}, {346, 24},  // (D24.15)
//  22  Blank fill
//  23  Carriage return
//  24  Bias for fifth band on this tape
    {0, 0}, {0, 0}, {401, 24},  // (D24.15)
//  25  Blank fill
//  26  Gain for fifth band on this tape
    {0, 0}, {426, 24},  // (D 9.15)
//  27  Blank fill
//  28  Carriage return
//  29  Bias for sixth band on this tape
    {0, 0}, {0, 0}, {481, 24},  // (D24.15)
//  30  Blank fill
//  31  Gain for sixth band on this tape
    {0, 0}, {506, 24},  // (D24.15)
//  32  Blank fill
//  33  Carriage return
//  34  Bias for seventh band on this tape
    {0, 0}, {0, 0}, {561, 24},  // (D24.15)
//  35  Blank fill
//  36  Gain for seventh band on this tape
    {0, 0}, {586, 24},  // (D24.15)
//  37  Blank fill
//  38  Carriage return
//  39  Bias for eighth band on this tape
```

```
    {0, 0}, {0, 0}, {641, 24},  // (D24.15)
//  40  Blank fill
//  41  Gain for eighth band on this tape
    {0, 0}, {666, 24},  // (D24.15)
//  42  Blank fill
//  43  Carriage return
//  44  Blank fill
//  45  Carriage return
//  46  "SENSORbGAINbSTATEb="
//  47  bbbnbbbnbbbnbbbnbbbnbbbnbbbnbbn
    {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {820, 32},  // (8*I4)
//  48  Blank fill
//  49  Carriage return
//  50  Blank fill
//  51  Carriage return
//  52  Blank fill
//  53  Carriage return
//  54  Blank fill
//  55  Carriage return
//  56  Blank fill
//  57  Carriage return
//  58  Blank fill
//  59  Carriage return
//  60  Blank fill
//  61  Carriage return
//  62  Blank fill
//  63  Carriage return
//  64  Blank fill
//  65  Carriage return
//  66  Blank fill
//  67  Carriage return
    {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0},
    {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0},
    {0, 0}, {0, 0}, {0, 0}, {0, 0}
};

////////////////////////////////////////////////////////////////////////////
// Geometric record fields

const int CFastFormat::m_iGRField[][2] =
{
//  To access by field name
    {0, 0},
//  1   "GEOMETRICbDATA"
//  2   "bMAPbPROJECTIONb="
//  3   Map projection name
    {0, 0}, {0, 0}, {32, 4},    //  (A4)
//  4   "bELLIPSOIDb="
//  5   Earth ellipsoid used
    {0, 0}, {48, 18},   //  (A18)
//  6   "bDATUMb="
//  7   Datum name
    {0, 0}, {74, 6},    //  (A6)
//  8   Carriage return
//  9   "USGSbPROJECTIONbPARAMETERSb="
    {0, 0}, {81, 28},   //  (A28)
//  10  Blank fill
//  11  USGS projection parameter #1: Semimajor axis
    {0, 0}, {110, 24},  //  (D24.15)
//  12  Blank fill
//  13  USGS projection parameter #2: Semiminor axis
    {0, 0}, {135, 24},  //  (D24.15)
```

```
//  14  Blank fill
//  15  Carriage return
//  16  USGS projection parameter #3
    {0, 0}, {0, 0}, {161, 24},  //  (D24.15)
//  17  Blank fill
//  18  USGS projection parameter #4
    {0, 0}, {186, 24},  //  (D24.15)
//  19  Blank fill
//  20  USGS projection parameter #5
    {0, 0}, {211, 24},  //  (D24.15)
//  21  Blank fill
//  22  Carriage return
//  23  USGS projection parameter #6
    {0, 0}, {0, 0}, {241, 24},  //  (D24.15)
//  24  Blank fill
//  25  USGS projection parameter #7
    {0, 0}, {266, 24},  //  (D24.15)
//  26  Blank fill
//  27  USGS projection parameter #8
    {0, 0}, {291, 24},  //  (D24.15)
//  28  Blank fill
//  29  Carriage return
//  30  USGS projection parameter #9
    {0, 0}, {0, 0}, {321, 24},  //  (D24.15)
//  31  Blank fill
//  32  USGS projection parameter #10
    {0, 0}, {346, 24},  //  (D24.15)
//  33  Blank fill
//  34  USGS projection parameter #11
    {0, 0}, {371, 24},  //  (D24.15)
//  35  Blank fill
//  36  Carriage return
//  37  USGS projection parameter #12
    {0, 0}, {0, 0}, {401, 24},  //  (D24.15)
//  38  Blank fill
//  39  USGS projection parameter #13
    {0, 0}, {426, 24},  //  (D24.15)
//  40  Blank fill
//  41  USGS projection parameter #14
    {0, 0}, {451, 24},  //  (D24.15)
//  42  Blank fill
//  43  Carriage return
//  44  USGS projection parameter #15
    {0, 0}, {0, 0}, {481, 24},  //  (D24.15)
//  45  Blank fill
//  46  Carriage return
//  47  "ULb="
//  48  Blank fill
//  49  Geodetic longitude of upper left corner of image (FIPS PUB 70)
    {0, 0}, {0, 0}, {0, 0}, {0, 0}, {566, 13},  //  (A13)
//  50  Blank fill
//  51  Geodetic latitude of upper left corner of image (FIPS PUB 70)
    {0, 0}, {580, 12},  //  (A12)
//  52  Blank fill
//  53  Easting of upper left corner of image in projections units
    {0, 0}, {593, 13},  //  (F13.3)
//  54  Blank fill
//  55  Northing of upper left corner of image in projection units
    {0, 0}, {607, 13},  //  (F13.3)
//  56  Blank fill
//  57  Carriage return
//  58  "URb="
```

```
//  59  Blank fill
//  60  Geodetic longitude of upper right corner of image (FIPS PUB 70)
    {0, 0}, {0, 0}, {0, 0}, {0, 0}, {646, 13},  //  (A13)
//  61  Blank fill
//  62  Geodetic latitude of upper right corner of image (FIPS PUB 70)
    {0, 0}, {660, 12},  //  (A12)
//  63  Blank fill
//  64  Easting of upper right corner of image in projections units
    {0, 0}, {673, 13},  //  (F13.3)
//  65  Blank fill
//  66  Northing of upper right corner of image in projection units
    {0, 0}, {687, 13},  //  (F13.3)
//  67  Blank fill
//  68  Carriage return
//  69  "LRb="
//  70  Blank fill
//  71  Geodetic longitude of lower right corner of image (FIPS PUB 70)
    {0, 0}, {0, 0}, {0, 0}, {0, 0}, {726, 13},  //  (A13)
//  72  Blank fill
//  73  Geodetic latitude of lower right corner of image (FIPS PUB 70)
    {0, 0}, {740, 12},  //  (A12)
//  74  Blank fill
//  75  Easting of lower right corner of image in projections units
    {0, 0}, {753, 13},  //  (F13.3)
//  76  Blank fill
//  77  Northing of lower right corner of image in projection units
    {0, 0}, {767, 13},  //  (F13.3)
//  78  Blank fill
//  79  Carriage return
//  80  "LLb="
//  81  Blank fill
//  82  Geodetic longitude of lower left corner of image (FIPS PUB 70)
    {0, 0}, {0, 0}, {0, 0}, {0, 0}, {806, 13},  //  (A13)
//  83  Blank fill
//  84  Geodetic latitude of lower left corner of image (FIPS PUB 70)
    {0, 0}, {820, 12},  //  (A12)
//  85  Blank fill
//  86  Easting of lower left corner of image in projections units
    {0, 0}, {833, 13},  //  (F13.3)
//  87  Blank fill
//  88  Northing of lower left corner of image in projection units
    {0, 0}, {847, 13},  //  (F13.3)
//  89  Blank fill
//  90  Carriage return
//  91  "CENTERb="
//  92  Blank fill
//  93  Scene center geodetic longitude (FIPS PUB 70)
    {0, 0}, {0, 0}, {0, 0}, {0, 0}, {890, 13},  //  (A13)
//  94  Blank fill
//  95  Scene center geodetic latitude (FIPS PUB 70)
    {0, 0}, {904, 12},  //  (A12)
//  96  Blank fill
//  97  Scene center easting in projection units
    {0, 0}, {917, 13},  //  (F13.3)
//  98  Blank fill
//  99  Scene center northing in projection units
    {0, 0}, {931, 13},  //  (F13.3)
//  100 Blank fill
//  101 Scene center pixel number measured left from the product
//      upper corner, rounded to nearest whole pixel (may be negative)
    {0, 0}, {945, 5},   //  (I5)
//  102 Blank fill
```

```
//  103 Scene center line number measured from the product left
//      corner upper rounded to nearest whole pixel (may be negative)
    {0, 0}, {951, 5},   //  (I5)
//  104 Blank fill
//  105 Carriage return
//  106 "OFFSETb="
//  107 Horizontal offset of the true scene center inunits of whole
//      pixels. (may be negative)
    {0, 0}, {0, 0}, {0, 0}, {969, 6},   //  (I6)
//  108 "bORIENTATIONbANGLEb="
//  109 Orientation angle in degrees (may be negative)
    {0, 0}, {995, 6},   //  (F6.2)
//  110 Blank fill
//  111 Carriage return
//  112 "SUNBbELEVATIONbANGLEb="
//  113 Sun elevation angle in degrees at scene center
    {0, 0}, {0, 0}, {0, 0}, {1062, 4},  //  (F4.1)
//  114 "bSUNbAZIMUTHbANGLEb="
//  115 Sun azimuth in degrees at scene center
    {0, 0}, {1086, 5},  //  (F5.1)
//  116 Blank fill
//  117 Carriage return
//  118 Blank fill
//  119 Carriage return
//  120 Blank fill
//  121 Carriage return
//  122 Blank fill
//  123 Carriage return
//  124 Blank fill
//  125 Carriage return
//  126 Blank fill
//  127 Carriage return
//  128 Blank fill
//  129 Carriage return
    {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0},
    {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}
};

////////////////////////////////////////////////////////////////////
// Construction/Destruction

/*
////////////////////////////////////////////////////////////////////
// Name:        CFastFormat
// Purpose:     Constructor
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
        const char* pFileName - Header file name
// Notes:
*/
CFastFormat::CFastFormat(const char* pFileName)
{
    m_pFile = new ifstream(pFileName, ios::in);
}

/*
////////////////////////////////////////////////////////////////////
// Name:        ~CFastFormat
// Purpose:     Destructor
// Context:     i42aresv Soft
// Returns:     void
```

```
// Parameters:
// Notes:
*/
CFastFormat::~CFastFormat()
{
    delete m_pFile;
}

////////////////////////////////////////////////////////////////////
// Operations

/*
////////////////////////////////////////////////////////////////////
// Name:        GetARField
// Purpose:     Gets a field of administrative record
// Context:     i42aresv Soft
// Returns:     char* - Field of administrative record
// Parameters:
        fstream* fHeader - Header file
        int iField - Field constant
// Notes:
*/
char* CFastFormat::GetARField(fstream* fHeader, int iField)
{
//  Get start byte and size
    int iStartByte = m_iARField[iField][StartByte];
    int iSize = m_iARField[iField][Size];

//  Get administrative record field
    char* pARField = new char[(iSize + 1)];
    fHeader->seekg((RecordSize * 0 + iStartByte - 1));
    fHeader->read(pARField, iSize);
    *(pARField + iSize) = '\0';

    return pARField;
}

/*
////////////////////////////////////////////////////////////////////
// Name:        GetRRField
// Purpose:     Gets a field of radiometric record
// Context:     i42aresv Soft
// Returns:     char* - Field of radiometric record
// Parameters:
        fstream* fHeader - Header file
        int iField - Field constant
// Notes:
*/
char* CFastFormat::GetRRField(fstream* fHeader, int iField)
{
//  Get start byte and size
    int iStartByte = m_iRRField[iField][StartByte];
    int iSize = m_iRRField[iField][Size];

//  Get radiometric record field
    char* pRRField = new char[(iSize + 1)];
    fHeader->seekg((RecordSize * 1 + iStartByte - 1));
    fHeader->read(pRRField, iSize);
    *(pRRField + iSize) = '\0';

    return pRRField;
}
```

```
/*
////////////////////////////////////////////////////////////////////
// Name:        GetGRField
// Purpose:     Gets a field of geometric record
// Context:     i42aresv Soft
// Returns:     char* - Field of geometric record
// Parameters:
        fstream* fHeader - Header file
        int iField - Field constant
// Notes:
*/
char* CFastFormat::GetGRField(fstream* fHeader, int iField)
{
//  Get start byte and size
    int iStartByte = m_iGRField[iField][StartByte];
    int iSize = m_iGRField[iField][Size];

//  Get geometric record field
    char* pGRField = new char[(iSize + 1)];
    fHeader->seekg((RecordSize * 2 + iStartByte - 1));
    fHeader->read(pGRField, iSize);
    *(pGRField + iSize) = '\0';

    return pGRField;
}

////////////////////////////////////////////////////////////////////
//  User functions to get file fields
```

IPSoft.h

```
// IPSoft.h: interface for the IPSoft classes.
//
////////////////////////////////////////////////////////////////////

#if !defined(AFX_IPSOFT_H_INCLUDED_)
#define AFX_IPSOFT_H_INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include <IPL.h>    //  Image Processing Library v2.5
#include <KLT.h>    //  KLT Library
#include <CProj.h>  //  GCTPC Library

#include "Image.h"
#include "MImage.h"
#include "DImage.h"
#include "Kernel.h"

#include "Matrix.h"     //  Matrix
#include "Polygon.h"    //  Polygon
#include "FastFormat.h" //  Header file (Fast Format Rev.C)

//  Additional header files
//

//  C++
#include <io.h> //  Additional functions
```

```
#include <iostream.h>   //  I/O functions (console)
#include <fstream.h>    //  I/O functions (file)

//  C
#include <stdio.h>      //  I/O functions
#include <stdlib.h>
#include <math.h>

/////////////////////////////////////////////////////////////////////
//  Macros

#define esc(a, b, c)    ((b) < (a) ? (a) : ((b) > (c) ? (c) : (b)))

//  TRACE function
#define TRACE(a)        {fprintf(stdout, a); fflush(stdout);}

#endif // !defined(AFX_IPSOFT_H_INCLUDED_)
```

A.3 Módulos

Los programas que hacen uso de la librería de clases e implementan cada una de las etapas propias del proceso de detección de cambios, son los siguientes:

1. `CorGeo`: Corrección geométrica
2. `CorRad`: Corrección radiométrica
3. `DifIma`: Diferencia de imágenes
4. `DetCam`: Detección de cambios (corrección geométrica y radiométrica, y diferencia de imágenes).

   NOTA: Como podemos observar, las etapas del proceso de detección de cambios se han implementado de forma modular para facilitar su incorporación a GIS disponibles en el mercado, como por ejemplo: GRASS (Geographic Resources Analysis Support System), etc.

A.3.1 Relación de ficheros

Los ficheros de código asociados a cada una de las etapas del proceso de detección de cambios, incluido el programa que realiza el proceso de forma completa, son los siguientes:

1. `CorGeo.cpp`
2. `CorRad.cpp`
3. `DifIma.cpp`
4. `DetCam.cpp`

CorGeo.cpp

```
// CorGeo.cpp : Defines the entry point for the console application.
//

#include <IPSoft.h>

//////////////////////////////////////////////////////////////////////
//  Parse functions

/*
//////////////////////////////////////////////////////////////////////
// Name:        getKeyArg
// Purpose:     Gets string from command line which corresponds to key
// Context:     i42aresv Soft
// Returns:     char* - Pointer to an argument string or NULL
// Parameters:
        int argc - Number of parameters
        char** argv - Parameters
        const char* key - Key for parameter
// Notes:
        Used by the getIntArg, getStrArg functions
*/
char* getKeyArg(int argc, char** argv, const char* key)
{
    char* p;
    for(int i = 1; i < argc; i++)
    {
        p = argv[i];
        if(p && strlen(p) >= 2 && ('/' == p[0] || '-' == p[0]) &&
                toupper(key[0]) == toupper(p[1]))
            return (p + 2);
    }
    return NULL;
}

/*
//////////////////////////////////////////////////////////////////////
// Name:        isKeyUsed
// Purpose:     Gets key from command line
// Context:     i42aresv Soft
// Returns:     bool - false as a rule
// Parameters:
        int argc - Number of parameters
        char** argv - Parameters
        const char* key - Key for parameter
// Notes:
*/
bool isKeyUsed(int argc, char** argv, const char* key)
{
```

```
    return (getKeyArg(argc, argv, key) != NULL);
}

/*
///////////////////////////////////////////////////////////////////////
// Name:        getIntArg
// Purpose:     Gets integer value from command line with a key
// Context:     i42ares Soft
// Returns:     bool - false as a rule
// Parameters:
        int argc - Number of parameters
        char** argv - Parameters
        const char* key - Key for parameter
        int* value - Value to be returned
// Notes:
        Use getKeyArg for string parameter getting
*/
bool getIntArg(int argc, char** argv, const char* key, int* value)
{
    char* p = getKeyArg(argc, argv, key);
    if(p && *p)
    {
        *value = atoi(p);
        return true;
    }
    return false;
}

/*
///////////////////////////////////////////////////////////////////////
// Name:        getStrArg
// Purpose:     Gets string value from command line with a key
// Context:     i42aresv Soft
// Returns:     bool - false as a rule
// Parameters:
        int argc - Number of parameters
        char** argv - Parameters
        const char* key - Key for parameter
        char* value - Value to be returned
// Notes:
        Use getKeyArg for string parameter getting
*/
bool getStrArg(int argc, char** argv, const char* key, char* value)
{
    char* p = getKeyArg(argc, argv, key);
    if(p && *p)
    {
        strcpy(value, p);
        return true;
    }
    return false;
}

/*
///////////////////////////////////////////////////////////////////////
// Name:        Usage
// Purpose:     Display program information
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
        const char* pFileName - File name
// Notes:
```

```
*/
void Usage(const char* pFileName)
{
    cerr << "Usage: " << pFileName << " -? -wWidth -hHeight -rRefImg \
        -cCorImg -tOutImg [-l] [-p] [-yPolygon]" << endl;
    cerr << "\t-?:\tDisplay this information" << endl;
    cerr << "\t-wWidth:\tImage width (Width % 128 = 0)" << endl;
    cerr << "\t-hHeight:\tImage height (Height % 128 = 0)" << endl;
    cerr << "\t-rRefImg:\tReference image" << endl;
    cerr << "\t-cCorImg:\tImage to correct" << endl;
    cerr << "\t-tOutImg:\tResult image" << endl;
    cerr << "\t-l:\tLog file" << endl;
    cerr << "\t-p:\tSave result image to PPM format" << endl;
    cerr << "\t-yPolygon:\tPolygon of interest" << endl;
}

/*
//////////////////////////////////////////////////////////////////////
// Name:        main
// Purpose:     Entry point
// Context:     i42aresv Soft
// Returns:     int - Return value
// Parameters:
        int argc - Number of parameters
        char* argv[] - Parameters
// Notes:
*/
int main(int argc, char* argv[])
{
//  Kernel
    CKernel CorGeo;

//////////////////////////////////////////////////////////////////////

    char* pFileName = "CorGeo";

    cout << pFileName << ":\tGeometric correction 1.0" << endl;
    cout << "(C) Copyright 2000-2001. Vicente M. Arevalo Espejo" << endl << endl;
    cout.flush();

    if(isKeyUsed(argc, argv, "?") == true)
    {
        Usage(pFileName);
        return 0;
    }

//  Number of arguments
    if(argc < 5)
    {
        Usage(pFileName);
        exit(-1);
    }

//  Image size
    int iWidth;
    int iHeight;

    if( (getIntArg(argc, argv, "w", &iWidth) == false) ||
        (getIntArg(argc, argv, "h", &iHeight) == false))
    {
        Usage(pFileName);
        exit(-1);
```

```
    }

//  IMPORTANT for tiling system
    if(((iWidth % 128) != 0) || ((iHeight % 128) != 0))
    {
        Usage(pFileName);
        exit(-1);
    }

//  Image files
    char pRefFileName[128];
    char pCorFileName[128];
    char pOutFileName[128];

//  File names
    if( (getStrArg(argc, argv, "r", pRefFileName) == false) ||
        (getStrArg(argc, argv, "c", pCorFileName) == false) ||
        (getStrArg(argc, argv, "t", pOutFileName) == false))
    {
        Usage(pFileName);
        exit(-1);
    }

//  Polygon file
    char pPolygonFileName[128];

//  File name
    if(getStrArg(argc, argv, "y", pPolygonFileName) == true)
        CorGeo.SetPolygonFile(pPolygonFileName);

    if(isKeyUsed(argc, argv, "l") == true)
        CorGeo.SetLog(true);

////////////////////////////////////////////////////////////////////////

//  Image objects (memory)
    CMImage RefImage(iWidth, iHeight);
    CMImage CorImage(iWidth, iHeight);
    CMImage OutImage(iWidth, iHeight);

    if(RefImage.LoadRAWFile(pRefFileName) == -1)
    {
        cerr << "ERROR <" << pFileName << ">: opening " << pRefFileName << endl;
        exit(-1);
    }

    if(CorImage.LoadRAWFile(pCorFileName) == -1)
    {
        cerr << "ERROR <" << pFileName << ">: opening " << pCorFileName << endl;
        exit(-1);
    }

//  Geometric correction (GPC)
    CorGeo.GeometricCorrection(&RefImage, &CorImage, &OutImage);

//  Save output image
    if(isKeyUsed(argc, argv, "p") == true)
        OutImage.SavePPMFile(pOutFileName);
    else
        OutImage.SaveRAWFile(pOutFileName);

    return 0;
```

```
}
```

CorRad.cpp

```
// CorRad.cpp : Defines the entry point for the console application.
//

#include <IPSoft.h>

/////////////////////////////////////////////////////////////////////////
//  Parse functions

/*
/////////////////////////////////////////////////////////////////////////
// Name:        getKeyArg
// Purpose:     Gets string from command line which corresponds to key
// Context:     i42aresv Soft
// Returns:     char* - Pointer to an argument string or NULL
// Parameters:
        int argc - Number of parameters
        char** argv - Parameters
        const char* key - Key for parameter
// Notes:
        Used by the getIntArg, getStrArg functions
*/
char* getKeyArg(int argc, char** argv, const char* key)
{
    char* p;
    for(int i = 1; i < argc; i++)
    {
        p = argv[i];
        if(p && strlen(p) >= 2 && ('/' == p[0] || '-' == p[0]) &&
                toupper(key[0]) == toupper(p[1]))
            return (p + 2);
    }
    return NULL;
}

/*
/////////////////////////////////////////////////////////////////////////
// Name:        isKeyUsed
// Purpose:     Gets key from command line
// Context:     i42aresv Soft
// Returns:     bool - false as a rule
// Parameters:
        int argc - Number of parameters
        char** argv - Parameters
        const char* key - Key for parameter
// Notes:
*/
bool isKeyUsed(int argc, char** argv, const char* key)
{
    return (getKeyArg(argc, argv, key) != NULL);
}

/*
/////////////////////////////////////////////////////////////////////////
// Name:        getIntArg
// Purpose:     Gets integer value from command line with a key
// Context:     i42ares Soft
// Returns:     bool - false as a rule
```

```
// Parameters:
        int argc - Number of parameters
        char** argv - Parameters
        const char* key - Key for parameter
        int* value - Value to be returned
// Notes:
        Use getKeyArg for string parameter getting
*/
bool getIntArg(int argc, char** argv, const char* key, int* value)
{
    char* p = getKeyArg(argc, argv, key);
    if(p && *p)
    {
        *value = atoi(p);
        return true;
    }
    return false;
}

/*
//////////////////////////////////////////////////////////////////////
// Name:        getStrArg
// Purpose:     Gets string value from command line with a key
// Context:     i42aresv Soft
// Returns:     bool - false as a rule
// Parameters:
        int argc - Number of parameters
        char** argv - Parameters
        const char* key - Key for parameter
        char* value - Value to be returned
// Notes:
        Use getKeyArg for string parameter getting
*/
bool getStrArg(int argc, char** argv, const char* key, char* value)
{
    char* p = getKeyArg(argc, argv, key);
    if(p && *p)
    {
        strcpy(value, p);
        return true;
    }
    return false;
}

/*
//////////////////////////////////////////////////////////////////////
// Name:        Usage
// Purpose:     Display program information
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
        const char* pFileName - File name
// Notes:
*/
void Usage(const char* pFileName)
{
    cerr << "Usage: " << pFileName << " -? -wWidth -hHeight -rRefImg \
        -cCorImg -tOutImg [-l] [-p]" << endl;
    cerr << "\t-?:\tDisplay this information" << endl;
    cerr << "\t-wWidth:\tImage width (Width % 128 = 0)" << endl;
    cerr << "\t-hHeight:\tImage height (Height % 128 = 0)" << endl;
    cerr << "\t-rRefImg:\tReference image" << endl;
```

```
    cerr << "\t-cCorImg:\tImage to correct" << endl;
    cerr << "\t-tOutImg:\tResult image" << endl;
    cerr << "\t-l:\tLog file" << endl;
    cerr << "\t-p:\tSave result image to PPM format" << endl;
}

/*
//////////////////////////////////////////////////////////////////////
// Name:        main
// Purpose:     Entry point
// Context:     i42aresv Soft
// Returns:     int - Return value
// Parameters:
        int argc - Number of parameters
        char* argv[] - Parameters
// Notes:
*/
int main(int argc, char* argv[])
{
//  Kernel
    CKernel CorRad;

//////////////////////////////////////////////////////////////////////

    char* pFileName = "CorRad";

    cout << pFileName << ":\tRadiometric correction 1.0" << endl;
    cout << "(C) Copyright 2000-2001. Vicente M. Arevalo Espejo" << endl << endl;
    cout.flush();

    if(isKeyUsed(argc, argv, "?") == true)
    {
        Usage(pFileName);
        return 0;
    }

//  Number of arguments
    if(argc < 5)
    {
        Usage(pFileName);
        exit(-1);
    }

//  Image size
    int iWidth;
    int iHeight;

    if( (getIntArg(argc, argv, "w", &iWidth) == false) ||
        (getIntArg(argc, argv, "h", &iHeight) == false))
    {
        Usage(pFileName);
        exit(-1);
    }

//  IMPORTANT for tiling system
    if(((iWidth % 128) != 0) || ((iHeight % 128) != 0))
    {
        Usage(pFileName);
        exit(-1);
    }

//  Image files
```

```
    char pRefFileName[128];
    char pCorFileName[128];
    char pOutFileName[128];

//  File names
    if( (getStrArg(argc, argv, "r", pRefFileName) == false) ||
        (getStrArg(argc, argv, "c", pCorFileName) == false) ||
        (getStrArg(argc, argv, "t", pOutFileName) == false))
    {
        Usage(pFileName);
        exit(-1);
    }

    if(isKeyUsed(argc, argv, "l") == true)
        CorRad.SetLog(true);

//////////////////////////////////////////////////////////////////////

//  Image objects (memory)
    CMImage RefImage(iWidth, iHeight);
    CMImage CorImage(iWidth, iHeight);
    CMImage OutImage(iWidth, iHeight);

    if(RefImage.LoadRAWFile(pRefFileName) == -1)
    {
        cerr << "ERROR <" << pFileName << ">: opening " << pRefFileName << endl;
        exit(-1);
    }

    if(CorImage.LoadRAWFile(pCorFileName) == -1)
    {
        cerr << "ERROR <" << pFileName << ">: opening " << pCorFileName << endl;
        exit(-1);
    }

//  Radiometric homogenization
    CorRad.RadiometricCorrection(&RefImage, &CorImage, &OutImage);

//  Save output image
    if(isKeyUsed(argc, argv, "p") == true)
        OutImage.SavePPMFile(pOutFileName);
    else
        OutImage.SaveRAWFile(pOutFileName);

    return 0;
}
```

DifIma.cpp

```
// DifIma.cpp : Defines the entry point for the console application.
//

#include <IPSoft.h>

//////////////////////////////////////////////////////////////////////
//  Parse functions

/*
//////////////////////////////////////////////////////////////////////
// Name:        getKeyArg
// Purpose:     Gets string from command line which corresponds to key
```

```
// Context:     i42aresv Soft
// Returns:     char* - Pointer to an argument string or NULL
// Parameters:
        int argc - Number of parameters
        char** argv - Parameters
        const char* key - Key for parameter
// Notes:
        Used by the getIntArg, getStrArg functions
*/
char* getKeyArg(int argc, char** argv, const char* key)
{
    char* p;
    for(int i = 1; i < argc; i++)
    {
        p = argv[i];
        if(p && strlen(p) >= 2 && ('/' == p[0] || '-' == p[0]) &&
                toupper(key[0]) == toupper(p[1]))
            return (p + 2);
    }
    return NULL;
}

/*
//////////////////////////////////////////////////////////////////////
// Name:        isKeyUsed
// Purpose:     Gets key from command line
// Context:     i42aresv Soft
// Returns:     bool - false as a rule
// Parameters:
        int argc - Number of parameters
        char** argv - Parameters
        const char* key - Key for parameter
// Notes:
*/
bool isKeyUsed(int argc, char** argv, const char* key)
{
    return (getKeyArg(argc, argv, key) != NULL);
}

/*
//////////////////////////////////////////////////////////////////////
// Name:        getIntArg
// Purpose:     Gets integer value from command line with a key
// Context:     i42ares Soft
// Returns:     bool - false as a rule
// Parameters:
        int argc - Number of parameters
        char** argv - Parameters
        const char* key - Key for parameter
        int* value - Value to be returned
// Notes:
        Use getKeyArg for string parameter getting
*/
bool getIntArg(int argc, char** argv, const char* key, int* value)
{
    char* p = getKeyArg(argc, argv, key);
    if(p && *p)
    {
        *value = atoi(p);
        return true;
    }
    return false;
```

```
}

/*
//////////////////////////////////////////////////////////////////////
// Name:        getStrArg
// Purpose:     Gets string value from command line with a key
// Context:     i42aresv Soft
// Returns:     bool - false as a rule
// Parameters:
        int argc - Number of parameters
        char** argv - Parameters
        const char* key - Key for parameter
        char* value - Value to be returned
// Notes:
        Use getKeyArg for string parameter getting
*/
bool getStrArg(int argc, char** argv, const char* key, char* value)
{
    char* p = getKeyArg(argc, argv, key);
    if(p && *p)
    {
        strcpy(value, p);
        return true;
    }
    return false;
}

/*
//////////////////////////////////////////////////////////////////////
// Name:        Usage
// Purpose:     Display program information
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
        const char* pFileName - File name
// Notes:
*/
void Usage(const char* pFileName)
{
    cerr << "Usage: " << pFileName << " -?|-a|-oOffset -wWidth -hHeight -rRefImg \
        -cCorImg -tOutImg [-l] [-p]" << endl;
    cerr << "\t-?:\tDisplay this information" << endl;
    cerr << "\t-a:\tAbsolute" << endl;
    cerr << "\t-oOffset:\tOffset" << endl;
    cerr << "\t-wWidth:\tImage width (Width % 128 = 0)" << endl;
    cerr << "\t-hHeight:\tImage height (Height % 128 = 0)" << endl;
    cerr << "\t-rRefImg:\tReference image" << endl;
    cerr << "\t-cCorImg:\tImage to correct" << endl;
    cerr << "\t-tOutImg:\tResult image" << endl;
    cerr << "\t-l:\tLog file" << endl;
    cerr << "\t-p:\tSave result image to PPM format" << endl;
}

/*
//////////////////////////////////////////////////////////////////////
// Name:        main
// Purpose:     Entry point
// Context:     i42aresv Soft
// Returns:     int - Return value
// Parameters:
        int argc - Number of parameters
        char* argv[] - Parameters
```

```
// Notes:
*/
int main(int argc, char* argv[])
{
//  Kernel
    CKernel DifIma;

////////////////////////////////////////////////////////////////////

    char* pFileName = "DifIma";

    cout << pFileName << ":\tImages difference 1.0" << endl;
    cout << "(C) Copyright 2000-2001. Vicente M. Arevalo Espejo" << endl << endl;
    cout.flush();

    if(isKeyUsed(argc, argv, "?") == true)
    {
        Usage(pFileName);
        return 0;
    }

//  Number of arguments
    if(argc < 6)
    {
        Usage(pFileName);
        exit(-1);
    }

//  Offset
    int iOffset;

//  Offset ? (Default value is Absolute)
    if(isKeyUsed(argc, argv, "a") == true)
    {
        DifIma.SetMode(CKernel::Absolute);
    }
    else
    {
        if(getIntArg(argc, argv, "o", &iOffset) == true)
        {
            DifIma.SetOffset(iOffset);
            DifIma.SetMode(CKernel::Offset);
        }
        else
        {
            Usage(pFileName);
            exit(-1);
        }
    }

//  Image size
    int iWidth;
    int iHeight;

    if( (getIntArg(argc, argv, "w", &iWidth) == false) ||
        (getIntArg(argc, argv, "h", &iHeight) == false))
    {
        Usage(pFileName);
        exit(-1);
    }

//  IMPORTANT for tiling system
```

```
    if(((iWidth % 128) != 0) || ((iHeight % 128) != 0))
    {
        Usage(pFileName);
        exit(-1);
    }

//  Image files
    char pRefFileName[128];
    char pCorFileName[128];
    char pOutFileName[128];

//  File names
    if( (getStrArg(argc, argv, "r", pRefFileName) == false) ||
        (getStrArg(argc, argv, "c", pCorFileName) == false) ||
        (getStrArg(argc, argv, "t", pOutFileName) == false))
    {
        Usage(pFileName);
        exit(-1);
    }

    if(isKeyUsed(argc, argv, "l") == true)
        DifIma.SetLog(true);

////////////////////////////////////////////////////////////////////////

//  Image objects (memory)
    CMImage RefImage(iWidth, iHeight);
    CMImage CorImage(iWidth, iHeight);
    CMImage OutImage(iWidth, iHeight);

    if(RefImage.LoadRAWFile(pRefFileName) == -1)
    {
        cerr << "ERROR <" << pFileName << ">: opening " << pRefFileName << endl;
        exit(-1);
    }

    if(CorImage.LoadRAWFile(pCorFileName) == -1)
    {
        cerr << "ERROR <" << pFileName << ">: opening " << pCorFileName << endl;
        exit(-1);
    }

//  Changes detection
    DifIma.ChangesDetection(&RefImage, &CorImage, &OutImage);

//  Save output image
    if(isKeyUsed(argc, argv, "p") == true)
        OutImage.SavePPMFile(pOutFileName);
    else
        OutImage.SaveRAWFile(pOutFileName);

    return 0;
}
```

DetCam.cpp

```
// DetCam.cpp : Defines the entry point for the application.
//

#include <IPSoft.h>
```

```
////////////////////////////////////////////////////////////////////
//  Parse functions

/*
////////////////////////////////////////////////////////////////////
// Name:        getKeyArg
// Purpose:     Gets string from command line which corresponds to key
// Context:     i42aresv Soft
// Returns:     char* - Pointer to an argument string or NULL
// Parameters:
        int argc - Number of parameters
        char** argv - Parameters
        const char* key - Key for parameter
// Notes:
        Used by the getIntArg, getStrArg functions
*/
char* getKeyArg(int argc, char** argv, const char* key)
{
    char* p;
    for(int i = 1; i < argc; i++)
    {
        p = argv[i];
        if(p && strlen(p) >= 2 && ('/' == p[0] || '-' == p[0]) &&
                toupper(key[0]) == toupper(p[1]))
            return (p + 2);
    }
    return NULL;
}

/*
////////////////////////////////////////////////////////////////////
// Name:        isKeyUsed
// Purpose:     Gets key from command line
// Context:     i42aresv Soft
// Returns:     bool - false as a rule
// Parameters:
        int argc - Number of parameters
        char** argv - Parameters
        const char* key - Key for parameter
// Notes:
*/
bool isKeyUsed(int argc, char** argv, const char* key)
{
    return (getKeyArg(argc, argv, key) != NULL);
}

/*
////////////////////////////////////////////////////////////////////
// Name:        getIntArg
// Purpose:     Gets integer value from command line with a key
// Context:     i42ares Soft
// Returns:     bool - false as a rule
// Parameters:
        int argc - Number of parameters
        char** argv - Parameters
        const char* key - Key for parameter
        int* value - Value to be returned
// Notes:
        Use getKeyArg for string parameter getting
*/
bool getIntArg(int argc, char** argv, const char* key, int* value)
{
```

```
    char* p = getKeyArg(argc, argv, key);
    if(p && *p)
    {
        *value = atoi(p);
        return true;
    }
    return false;
}

/*
/////////////////////////////////////////////////////////////////////
// Name:        getStrArg
// Purpose:     Gets string value from command line with a key
// Context:     i42aresv Soft
// Returns:     bool - false as a rule
// Parameters:
        int argc - Number of parameters
        char** argv - Parameters
        const char* key - Key for parameter
        char* value - Value to be returned
// Notes:
        Use getKeyArg for string parameter getting
*/
bool getStrArg(int argc, char** argv, const char* key, char* value)
{
    char* p = getKeyArg(argc, argv, key);
    if(p && *p)
    {
        strcpy(value, p);
        return true;
    }
    return false;
}

/*
/////////////////////////////////////////////////////////////////////
// Name:        Usage
// Purpose:     Display program information
// Context:     i42aresv Soft
// Returns:     void
// Parameters:
        const char* pFileName - File name
// Notes:
*/
void Usage(const char* pFileName)
{
    cerr << "Usage: " << pFileName << " -?|-a|-oOffset -wWidth -hHeight -rRefImg \
        -cCorImg -tOutImg [-l] [-i] [-p] [-yPolygon]" << endl;
    cerr << "\t-?:\tDisplay this information" << endl;
    cerr << "\t-a:\tAbsolute (Images difference)" << endl;
    cerr << "\t-oOffset:\tOffset (Images difference)" << endl;
    cerr << "\t-wWidth:\tImage width (Width % 128 = 0)" << endl;
    cerr << "\t-hHeight:\tImage height (Height % 128 = 0)" << endl;
    cerr << "\t-rRefImg:\tReference image" << endl;
    cerr << "\t-cCorImg:\tImage to correct" << endl;
    cerr << "\t-tOutImg:\tResult image" << endl;
    cerr << "\t-l:\tLog file" << endl;
    cerr << "\t-i:\tIntermediate images" << endl;
    cerr << "\t-p:\tSave result image to PPM format" << endl;
    cerr << "\t-yPolygon:\tPolygon of interest (Geometric correction)" << endl;
}
```

```
/*
//////////////////////////////////////////////////////////////////////
// Name:        main
// Purpose:     Entry point
// Context:     i42aresv Soft
// Returns:     int - Return value
// Parameters:
        int argc - Number of parameters
        char* argv[] - Parameters
// Notes:
*/
int main(int argc, char* argv[])
{
//  Kernel
    CKernel DetCam;

//////////////////////////////////////////////////////////////////////

    char* pFileName = "DetCam";

    cout << pFileName << ":\tChanges detection 1.0" << endl;
    cout << "(C) Copyright 2000-2001. Vicente M. Arevalo Espejo" << endl << endl;
    cout.flush();

    if(isKeyUsed(argc, argv, "?") == true)
    {
        Usage(pFileName);
        return 0;
    }

//  Number of arguments
    if(argc < 6)
    {
        Usage(pFileName);
        exit(-1);
    }

//  Offset
    int iOffset;

//  Offset ? (Default value is Absolute)
    if(isKeyUsed(argc, argv, "a") == true)
        DetCam.SetMode(CKernel::Absolute);
    else
    {
        if(getIntArg(argc, argv, "o", &iOffset) == true)
        {
            DetCam.SetOffset(iOffset);
            DetCam.SetMode(CKernel::Offset);
        }
        else
        {
            Usage(pFileName);
            exit(-1);
        }
    }

//  Image size
    int iWidth;
    int iHeight;

    if( (getIntArg(argc, argv, "w", &iWidth) == false) ||
```

```
        (getIntArg(argc, argv, "h", &iHeight) == false))
    {
        Usage(pFileName);
        exit(-1);
    }

//  IMPORTANT for tiling system
    if(((iWidth % 128) != 0) || ((iHeight % 128) != 0))
    {
        Usage(pFileName);
        exit(-1);
    }

//  Image files
    char pRefFileName[128];
    char pCorFileName[128];
    char pOutFileName[128];

//  File names
    if( (getStrArg(argc, argv, "r", pRefFileName) == false) ||
        (getStrArg(argc, argv, "c", pCorFileName) == false) ||
        (getStrArg(argc, argv, "t", pOutFileName) == false))
    {
        Usage(pFileName);
        exit(-1);
    }

//  Polygon file
    char pPolygonFileName[128];

//  File name
    if(getStrArg(argc, argv, "y", pPolygonFileName) == true)
        DetCam.SetPolygonFile(pPolygonFileName);

    if(isKeyUsed(argc, argv, "l") == true)
        DetCam.SetLog(true);

////////////////////////////////////////////////////////////////////////

//  Image objects (memory)
    CMImage RefImage(iWidth, iHeight);
    CMImage CorImage(iWidth, iHeight);
    CMImage OutImage(iWidth, iHeight);

    if(RefImage.LoadRAWFile(pRefFileName) == -1)
    {
        cerr << "ERROR <" << pFileName << ">: opening " << pRefFileName << endl;
        exit(-1);
    }

    if(CorImage.LoadRAWFile(pCorFileName) == -1)
    {
        cerr << "ERROR <" << pFileName << ">: opening " << pCorFileName << endl;
        exit(-1);
    }

    bool bImages = isKeyUsed(argc, argv, "i");

//  Geometric correction (GPC)
    DetCam.GeometricCorrection(&RefImage, &CorImage, &OutImage);

    if(bImages == true)
```

```
        OutImage.SaveRAWFile("GeoCor.raw");

//  Radiometric homogenization
    DetCam.RadiometricCorrection(&RefImage, &OutImage, &CorImage);

    if(bImages == true)
        CorImage.SaveRAWFile("RadCor.raw");

//  Changes detection
    DetCam.ChangesDetection(&RefImage, &CorImage, &OutImage);

//  Save output image
    if(isKeyUsed(argc, argv, "p") == true)
        OutImage.SavePPMFile(pOutFileName);
    else
        OutImage.SaveRAWFile(pOutFileName);

    return 0;
}
```

A.4 Interfaz gráfico

El interfaz que gestiona la ejecución de los programas, y visualiza los resultados obtenidos es el siguiente:

1. `DetCam`: Corrección geométrica y radiométrica, diferencia de imágenes y detección de cambios. Visualización de los resultados obtenidos:
   (a) GCPs detectados, eliminados, etc.
   (b) Imágenes resultado e intermedias (corrección geométrica y radiométrica)

A.4.1 Relación de ficheros

El fichero de código asociado al interfaz gráfico es el siguiente:

1. `DetCam.tcl`

   NOTA: Requiere TCL/TK 8.3 para Windows[1].

DetCam.tcl

```
#!/bin/sh
# the next line restarts using wish\
exec wish "$0" "$@"
```

[1] `http://dev.scriptics.com/software/tcltk/`

```
if {![info exists vTcl(sourcing)]} {
    switch $tcl_platform(platform) {
    windows {
    }
    default {
        option add *Scrollbar.width 10
    }
    }

}
#############################################################################
# Visual Tcl v1.51 Project
#

#################################
# VTCL LIBRARY PROCEDURES
#

if {![info exists vTcl(sourcing)]} {
proc Window {args} {
    global vTcl
    set cmd     [lindex $args 0]
    set name    [lindex $args 1]
    set newname [lindex $args 2]
    set rest    [lrange $args 3 end]
    if {$name == "" || $cmd == ""} { return }
    if {$newname == ""} { set newname $name }
    if {$name == "."} { wm withdraw $name; return }
    set exists [winfo exists $newname]
    switch $cmd {
        show {
            if {$exists} { wm deiconify $newname; return }
            if {[info procs vTclWindow(pre)$name] != ""} {
                eval "vTclWindow(pre)$name $newname $rest"
            }
            if {[info procs vTclWindow$name] != ""} {
                eval "vTclWindow$name $newname $rest"
            }
            if {[info procs vTclWindow(post)$name] != ""} {
                eval "vTclWindow(post)$name $newname $rest"
            }
        }
        hide    { if $exists {wm withdraw $newname; return} }
        iconify { if $exists {wm iconify $newname; return} }
        destroy { if $exists {destroy $newname; return} }
    }
}

proc vTcl:WindowsCleanup {} {
    global vTcl
    if {[info exists vTcl(sourcing)]} { return }
    foreach w [winfo children .] {
        wm protocol $w WM_DELETE_WINDOW { exit }
    }
}
}

if {![info exists vTcl(sourcing)]} {
proc {vTcl:DefineAlias} {target alias widgetProc top_or_alias cmdalias} {
    global widget
```

```
        set widget($alias) $target
        set widget(rev,$target) $alias

        if {$cmdalias} {
            interp alias {} $alias {} $widgetProc $target
        }

        if {$top_or_alias != ""} {
            set widget($top_or_alias,$alias) $target

            if {$cmdalias} {
                interp alias {} $top_or_alias.$alias {} $widgetProc $target
            }
        }
    }

    proc {vTcl:Toplevel:WidgetProc} {w args} {
        if {[llength $args] == 0} {
            return -code error "wrong # args: should be \"$w option ?arg arg ...?\""
        }

        ## The first argument is a switch, they must be doing a configure.
        if {[string index $args 0] == "-"} {
            set command configure

            ## There's only one argument, must be a cget.
            if {[llength $args] == 1} {
                set command cget
            }
        } else {
            set command [lindex $args 0]
            set args [lrange $args 1 end]
        }

        switch -- $command {
            "hide" -
            "Hide" {
                Window hide $w
            }

            "show" -
            "Show" {
                Window show $w
            }

            "ShowModal" {
                Window show $w
                raise $w
                grab $w
                tkwait window $w
                grab release $w
            }

            default {
                eval $w $command $args
            }
        }
    }

    proc {vTcl:WidgetProc} {w args} {
        if {[llength $args] == 0} {
            return -code error "wrong # args: should be \"$w option ?arg arg ...?\""
```

```
    }

    ## The first argument is a switch, they must be doing a configure.
    if {[string index $args 0] == "-"} {
        set command configure

        ## There's only one argument, must be a cget.
        if {[llength $args] == 1} {
            set command cget
        }
    } else {
        set command [lindex $args 0]
        set args [lrange $args 1 end]
    }

    eval $w $command $args
}
}

if {[info exists vTcl(sourcing)]} {
proc vTcl:project:info {} {
    namespace eval ::widgets::.top17 {
        array set save {-menu 1}
    }
    namespace eval ::widgets::.top17.cpd30 {
        array set save {-borderwidth 1 -height 1 -relief 1 -width 1}
    }
    namespace eval ::widgets::.top17.cpd30.01 {
        array set save {-anchor 1 -borderwidth 1 -menu 1 -padx 1 -pady 1 -text 1 \
            -underline 1 -width 1}
    }
    namespace eval ::widgets::.top17.cpd30.01.02 {
        array set save {-activeborderwidth 1 -borderwidth 1 -cursor 1}
    }
    namespace eval ::widgets::.top17.cpd30.05 {
        array set save {-anchor 1 -borderwidth 1 -menu 1 -padx 1 -pady 1 -text 1 \
            -underline 1 -width 1}
    }
    namespace eval ::widgets::.top17.cpd30.05.06 {
        array set save {-activeborderwidth 1 -borderwidth 1 -cursor 1}
    }
    namespace eval ::widgets::.top17.fra17 {
        array set save {-borderwidth 1 -height 1 -relief 1 -width 1}
    }
    namespace eval ::widgets::.top17.fra17.fra17 {
        array set save {-borderwidth 1 -cursor 1 -height 1 -relief 1 -width 1}
    }
    namespace eval ::widgets::.top17.fra17.fra17.ent17 {
        array set save {-textvariable 1}
    }
    namespace eval ::widgets::.top17.fra17.fra17.but17 {
        array set save {-command 1 -pady 1 -text 1}
    }
    namespace eval ::widgets::.top17.fra17.lab17 {
        array set save {-borderwidth 1 -text 1}
    }
    namespace eval ::widgets::.top17.fra17.fra18 {
        array set save {-borderwidth 1 -cursor 1 -height 1 -relief 1 -width 1}
    }
    namespace eval ::widgets::.top17.fra17.fra18.ent17 {
        array set save {-textvariable 1}
    }
```

```
namespace eval ::widgets::.top17.fra17.fra18.but17 {
    array set save {-command 1 -pady 1 -text 1}
}
namespace eval ::widgets::.top17.fra17.lab18 {
    array set save {-borderwidth 1 -text 1}
}
namespace eval ::widgets::.top17.lab17 {
    array set save {-borderwidth 1 -text 1}
}
namespace eval ::widgets::.top17.fra18 {
    array set save {-borderwidth 1 -height 1 -relief 1 -width 1}
}
namespace eval ::widgets::.top17.fra18.fra17 {
    array set save {-borderwidth 1 -cursor 1 -height 1 -relief 1 -width 1}
}
namespace eval ::widgets::.top17.fra18.fra17.ent17 {
    array set save {-textvariable 1}
}
namespace eval ::widgets::.top17.fra18.fra17.but17 {
    array set save {-command 1 -pady 1 -text 1}
}
namespace eval ::widgets::.top17.fra18.lab17 {
    array set save {-borderwidth 1 -text 1}
}
namespace eval ::widgets::.top17.lab18 {
    array set save {-borderwidth 1 -text 1}
}
namespace eval ::widgets::.top17.fra19 {
    array set save {-borderwidth 1 -height 1 -relief 1 -width 1}
}
namespace eval ::widgets::.top17.fra19.fra17 {
    array set save {-borderwidth 1 -cursor 1 -height 1 -relief 1 -width 1}
}
namespace eval ::widgets::.top17.fra19.fra17.lab17 {
    array set save {-borderwidth 1 -text 1}
}
namespace eval ::widgets::.top17.fra19.fra17.lab18 {
    array set save {-borderwidth 1 -text 1}
}
namespace eval ::widgets::.top17.fra19.fra17.ent17 {
    array set save {-textvariable 1}
}
namespace eval ::widgets::.top17.fra19.fra17.ent18 {
    array set save {-textvariable 1}
}
namespace eval ::widgets::.top17.fra19.lab17 {
    array set save {-borderwidth 1 -text 1}
}
namespace eval ::widgets::.top17.fra19.che17 {
    array set save {-text 1 -underline 1 -variable 1}
}
namespace eval ::widgets::.top17.fra19.che18 {
    array set save {-text 1 -underline 1 -variable 1}
}
namespace eval ::widgets::.top17.lab19 {
    array set save {-borderwidth 1 -text 1}
}
namespace eval ::widgets::.top18 {
    array set save {}
}
namespace eval ::widgets::.top18.but18 {
    array set save {-command 1 -pady 1 -text 1 -underline 1}
```

```
}
namespace eval ::widgets::.top18.but19 {
    array set save {-command 1 -pady 1 -state 1 -text 1 -underline 1}
}
namespace eval ::widgets::.top18.but20 {
    array set save {-command 1 -pady 1 -text 1 -underline 1}
}
namespace eval ::widgets::.top18.fra18 {
    array set save {-borderwidth 1 -cursor 1 -height 1 -relief 1 -width 1}
}
namespace eval ::widgets::.top18.fra18.che18 {
    array set save {-command 1 -text 1 -underline 1 -variable 1}
}
namespace eval ::widgets::.top18.fra18.ent18 {
    array set save {-state 1 -textvariable 1}
}
namespace eval ::widgets::.top18.fra18.but18 {
    array set save {-command 1 -pady 1 -state 1 -text 1}
}
namespace eval ::widgets::.top18.fra19 {
    array set save {-borderwidth 1 -height 1 -relief 1 -width 1}
}
namespace eval ::widgets::.top18.fra19.tex18 {
    array set save {-yscrollcommand 1}
}
namespace eval ::widgets::.top18.fra19.scr18 {
    array set save {-command 1}
}
namespace eval ::widgets::.top18.lab18 {
    array set save {-borderwidth 1 -text 1}
}
namespace eval ::widgets::.top19 {
    array set save {-cursor 1}
}
namespace eval ::widgets::.top19.but19 {
    array set save {-command 1 -pady 1 -text 1 -underline 1}
}
namespace eval ::widgets::.top19.but20 {
    array set save {-command 1 -pady 1 -state 1 -text 1 -underline 1}
}
namespace eval ::widgets::.top19.but21 {
    array set save {-command 1 -pady 1 -text 1 -underline 1}
}
namespace eval ::widgets::.top19.fra19 {
    array set save {-borderwidth 1 -height 1 -relief 1 -width 1}
}
namespace eval ::widgets::.top19.fra19.tex19 {
    array set save {-yscrollcommand 1}
}
namespace eval ::widgets::.top19.fra19.scr19 {
    array set save {-command 1}
}
namespace eval ::widgets::.top20 {
    array set save {}
}
namespace eval ::widgets::.top20.but20 {
    array set save {-command 1 -pady 1 -text 1 -underline 1}
}
namespace eval ::widgets::.top20.but21 {
    array set save {-command 1 -pady 1 -state 1 -text 1 -underline 1}
}
namespace eval ::widgets::.top20.but22 {
```

```
    array set save {-command 1 -pady 1 -text 1 -underline 1}
}
namespace eval ::widgets::.top20.fra20 {
    array set save {-borderwidth 1 -height 1 -relief 1 -width 1}
}
namespace eval ::widgets::.top20.fra20.rad20 {
    array set save {-command 1 -text 1 -value 1 -variable 1}
}
namespace eval ::widgets::.top20.fra20.rad21 {
    array set save {-command 1 -text 1 -value 1 -variable 1}
}
namespace eval ::widgets::.top20.fra20.ent20 {
    array set save {-justify 1 -state 1 -textvariable 1}
}
namespace eval ::widgets::.top20.fra20.lab20 {
    array set save {-borderwidth 1 -state 1 -text 1}
}
namespace eval ::widgets::.top20.fra21 {
    array set save {-borderwidth 1 -height 1 -relief 1 -width 1}
}
namespace eval ::widgets::.top20.fra21.tex20 {
    array set save {-yscrollcommand 1}
}
namespace eval ::widgets::.top20.fra21.scr20 {
    array set save {-command 1}
}
namespace eval ::widgets::.top20.lab20 {
    array set save {-borderwidth 1 -text 1}
}
namespace eval ::widgets::.top21 {
    array set save {}
}
namespace eval ::widgets::.top21.but21 {
    array set save {-command 1 -pady 1 -text 1 -underline 1}
}
namespace eval ::widgets::.top21.but22 {
    array set save {-command 1 -pady 1 -text 1 -underline 1}
}
namespace eval ::widgets::.top21.but23 {
    array set save {-command 1 -pady 1 -text 1 -underline 1}
}
namespace eval ::widgets::.top21.fra21 {
    array set save {-borderwidth 1 -height 1 -relief 1 -width 1}
}
namespace eval ::widgets::.top21.fra21.rad21 {
    array set save {-command 1 -text 1 -value 1 -variable 1}
}
namespace eval ::widgets::.top21.fra21.rad22 {
    array set save {-command 1 -text 1 -value 1 -variable 1}
}
namespace eval ::widgets::.top21.fra21.ent21 {
    array set save {-justify 1 -state 1 -textvariable 1}
}
namespace eval ::widgets::.top21.fra21.lab21 {
    array set save {-borderwidth 1 -state 1 -text 1}
}
namespace eval ::widgets::.top21.fra21.che21 {
    array set save {-command 1 -text 1 -underline 1 -variable 1}
}
namespace eval ::widgets::.top21.fra21.ent22 {
    array set save {-textvariable 1}
}
```

```
    namespace eval ::widgets::.top21.fra21.but21 {
        array set save {-command 1 -pady 1 -text 1}
    }
    namespace eval ::widgets::.top21.fra21.che22 {
        array set save {-justify 1 -text 1 -underline 1 -variable 1}
    }
    namespace eval ::widgets::.top21.fra22 {
        array set save {-borderwidth 1 -height 1 -relief 1 -width 1}
    }
    namespace eval ::widgets::.top21.fra22.tex21 {
        array set save {-yscrollcommand 1}
    }
    namespace eval ::widgets::.top21.fra22.scr21 {
        array set save {-command 1}
    }
    namespace eval ::widgets::.top21.lab21 {
        array set save {-borderwidth 1 -text 1}
    }
    namespace eval ::widgets::.top22 {
        array set save {-cursor 1}
    }
    namespace eval ::widgets::.top22.scr22 {
        array set save {-command 1}
    }
    namespace eval ::widgets::.top22.tex22 {
        array set save {-height 1 -wrap 1 -yscrollcommand 1}
    }
    namespace eval ::widgets::.top23 {
        array set save {}
    }
    namespace eval ::widgets::.top23.lab23 {
        array set save {-borderwidth 1}
    }
    namespace eval ::widgets_bindings {
        set tagslist {}
    }
}
}
###################################
# USER DEFINED PROCEDURES
#

proc {main} {argc argv} {
wm protocol .top17 WM_DELETE_WINDOW {exit}
}

proc init {argc argv} {

}

init $argc $argv

###################################
# VTCL GENERATED GUI PROCEDURES
#

proc vTclWindow. {base {container 0}} {
    if {$base == ""} {
        set base .
    }
    ###################
    # CREATING WIDGETS
```

```
    ####################
    if {!$container} {
    wm focusmodel $base passive
    wm geometry $base 100x100+0+0; update
    wm maxsize $base 200 200
    wm minsize $base 200 200
    wm overrideredirect $base 0
    wm resizable $base 0 0
    wm withdraw $base
    wm title $base "vt"
    bindtags $base "$base Vtcl all"
    }
    ####################
    # SETTING GEOMETRY
    ####################
}

proc vTclWindow.top17 {base {container 0}} {
    if {$base == ""} {
        set base .top17
    }
    if {[winfo exists $base] && (!$container)} {
        wm deiconify $base; return
    }

    global widget

    ####################
    # CREATING WIDGETS
    ####################
    if {!$container} {
    toplevel $base -class Toplevel \
        -menu "$base.m50"
    wm focusmodel $base passive
    wm geometry $base 400x420+100+100; update
    wm maxsize $base 1024 768
    wm minsize $base 106 2
    wm overrideredirect $base 0
    wm resizable $base 0 0
    wm deiconify $base
    wm title $base "DetCam Interface 1.0"
    }
    frame $base.cpd30 \
        -borderwidth 1 -height 30 -relief sunken -width 30
    menubutton $base.cpd30.01 \
        -anchor w -borderwidth 3 -menu "$base.cpd30.01.02" -padx 4 -pady 3 \
        -text Program -underline 0 -width 6
    menu $base.cpd30.01.02 \
        -activeborderwidth 1 -borderwidth 1 -cursor {}
    $base.cpd30.01.02 add command \
        -accelerator {} \
        -command {set bPolygon 0
Window show .top18
focus .top18
.top18.fra19.tex18 delete 0.0 [.top18.fra19.tex18 index end]
.top18.fra19.tex18 insert 0.0 "Push Run button"
.top18.but19 configure -state disable} \
        -image {} -label {Geometric correction}
    $base.cpd30.01.02 add command \
        -accelerator {} \
        -command {Window show .top19
focus .top19
```

```
.top19.fra19.tex19 delete 0.0 [.top19.fra19.tex19 index end]
.top19.fra19.tex19 insert 0.0 "Push Run button"
.top19.but20 configure -state disable} \
        -image {} -label {Radiometric correction}
    $base.cpd30.01.02 add command \
        -accelerator {} \
        -command {set iMode 0
set iOffset 127
Window show .top20
focus .top20
.top20.fra20.ent20 configure -state disable
.top20.fra20.lab20 configure -state disable
.top20.fra21.tex20 delete 0.0 [.top20.fra21.tex20 index end]
.top20.fra21.tex20 insert 0.0 "Push Run button"
.top20.but21 configure -state disable} \
        -image {} -label {Images difference}
    $base.cpd30.01.02 add separator
    $base.cpd30.01.02 add command \
        -accelerator {} \
        -command {set iMode 0
set iOffset 127
set bPolygon 0
set bImages 1
Window show .top21
focus .top21
.top21.fra21.ent21 configure -state disable
.top21.fra21.lab21 configure -state disable
.top21.fra22.tex21 delete 0.0 [.top21.fra22.tex21 index end]
.top21.fra22.tex21 insert 0.0 "Push Run button"
.top21.but22 configure -state disable} \
        -image {} -label {Changes detection}
    menubutton $base.cpd30.05 \
        -anchor w -borderwidth 3 -menu "$base.cpd30.05.06" -padx 4 -pady 3 \
        -text Help -underline 0 -width 4
    menu $base.cpd30.05.06 \
        -activeborderwidth 1 -borderwidth 1 -cursor {}
    $base.cpd30.05.06 add command \
        -accelerator {} \
        -command tk_messageBox -title "About DetCam ..." \
            -message "The programs CorGeo.exe, CorRad.exe, DifIma.exe, DetCam.exe \
                and DetCam Interface 1.0\nare copyrighted by i42aresv." -type ok \
        -image {} -label {About DetCam ...}
    frame $base.fra17 \
        -borderwidth 2 -height 75 -relief groove -width 125
    frame $base.fra17.fra17 \
        -borderwidth 2 -cursor arrow -height 75 -relief groove -width 125
    entry $base.fra17.fra17.ent17 \
        -textvariable imgRImage
    button $base.fra17.fra17.but17 \
        \
        -command {set types { {{RAW files} {.RAW}} {{All files} *} }
set imgRImage [tk_getOpenFile -filetypes $types]} \
        -pady 0 -text <<
    label $base.fra17.lab17 \
        -borderwidth 1 -text {Reference image (RAW)}
    frame $base.fra17.fra18 \
        -borderwidth 2 -cursor arrow -height 75 -relief groove -width 125
    entry $base.fra17.fra18.ent17 \
        -textvariable imgCImage
    button $base.fra17.fra18.but17 \
        \
        -command {set types { {{RAW files} {.RAW}} {{All files} *} }
```

```
set imgCImage [tk_getOpenFile -filetypes $types]} \
        -pady 0 -text <<
    label $base.fra17.lab18 \
        -borderwidth 1 -text {Image to correct (RAW)}
    label $base.lab17 \
        -borderwidth 1 -text Input
    frame $base.fra18 \
        -borderwidth 2 -height 75 -relief groove -width 125
    frame $base.fra18.fra17 \
        -borderwidth 2 -cursor arrow -height 75 -relief groove -width 125
    entry $base.fra18.fra17.ent17 \
        -textvariable imgOImage
    button $base.fra18.fra17.but17 \
        \
        -command {set imgOImage [tk_chooseDirectory -mustexist true]
if { [string compare $imgOImage ""] != 0 } { \
    append imgOImage "/Result.RAW" \
} \
        -pady 0 -text <<
    label $base.fra18.lab17 \
        -borderwidth 1 -text {Result image (RAW)}
    label $base.lab18 \
        -borderwidth 1 -text Output
    frame $base.fra19 \
        -borderwidth 2 -height 75 -relief groove -width 125
    frame $base.fra19.fra17 \
        -borderwidth 2 -cursor arrow -height 75 -relief groove -width 125
    label $base.fra19.fra17.lab17 \
        -borderwidth 1 -text Width
    label $base.fra19.fra17.lab18 \
        -borderwidth 1 -text Height
    entry $base.fra19.fra17.ent17 \
        -textvariable iWidth
    entry $base.fra19.fra17.ent18 \
        -textvariable iHeight
    label $base.fra19.lab17 \
        -borderwidth 1 -text {Image size}
    checkbutton $base.fra19.che17 \
        -text {Generate log file} -underline 0 -variable bLog
    checkbutton $base.fra19.che18 \
        -text {Save result image to PPM} -underline 0 -variable bPPM
    label $base.lab19 \
        -borderwidth 1 -text {Additional arguments}
    ####################
    # SETTING GEOMETRY
    ####################
    place $base.cpd30 \
        -x 10 -y 7 -width 380 -height 25 -anchor nw -bordermode ignore
    pack $base.cpd30.01 \
        -in $base.cpd30 -anchor center -expand 0 -fill none -side left
    pack $base.cpd30.05 \
        -in $base.cpd30 -anchor center -expand 0 -fill none -side right
    place $base.fra17 \
        -x 9 -y 48 -width 380 -height 135 -anchor nw -bordermode ignore
    place $base.fra17.fra17 \
        -x 12 -y 16 -width 355 -height 45 -anchor nw -bordermode ignore
    place $base.fra17.fra17.ent17 \
        -x 13 -y 15 -width 286 -height 19 -anchor nw -bordermode ignore
    place $base.fra17.fra17.but17 \
        -x 311 -y 15 -width 30 -height 18 -anchor nw -bordermode ignore
    place $base.fra17.lab17 \
        -x 20 -y 8 -width 121 -height 17 -anchor nw -bordermode ignore
```

```
    place $base.fra17.fra18 \
        -x 12 -y 76 -width 355 -height 45 -anchor nw -bordermode ignore
    place $base.fra17.fra18.ent17 \
        -x 13 -y 15 -width 286 -height 19 -anchor nw -bordermode ignore
    place $base.fra17.fra18.but17 \
        -x 311 -y 15 -width 30 -height 18 -anchor nw -bordermode ignore
    place $base.fra17.lab18 \
        -x 20 -y 68 -width 116 -height 17 -anchor nw -bordermode ignore
    place $base.lab17 \
        -x 15 -y 40 -width 31 -height 17 -anchor nw -bordermode ignore
    place $base.fra18 \
        -x 9 -y 198 -width 380 -height 75 -anchor nw -bordermode ignore
    place $base.fra18.fra17 \
        -x 12 -y 16 -width 355 -height 45 -anchor nw -bordermode ignore
    place $base.fra18.fra17.ent17 \
        -x 13 -y 15 -width 286 -height 19 -anchor nw -bordermode ignore
    place $base.fra18.fra17.but17 \
        -x 311 -y 15 -width 30 -height 18 -anchor nw -bordermode ignore
    place $base.fra18.lab17 \
        -x 20 -y 8 -width 101 -height 17 -anchor nw -bordermode ignore
    place $base.lab18 \
        -x 15 -y 190 -width 36 -height 17 -anchor nw -bordermode ignore
    place $base.fra19 \
        -x 9 -y 288 -width 380 -height 120 -anchor nw -bordermode ignore
    place $base.fra19.fra17 \
        -x 12 -y 41 -width 355 -height 65 -anchor nw -bordermode ignore
    place $base.fra19.fra17.lab17 \
        -x 13 -y 13 -width 32 -height 17 -anchor nw -bordermode ignore
    place $base.fra19.fra17.lab18 \
        -x 187 -y 13 -width 36 -height 17 -anchor nw -bordermode ignore
    place $base.fra19.fra17.ent17 \
        -x 15 -y 32 -width 151 -height 19 -anchor nw -bordermode ignore
    place $base.fra19.fra17.ent18 \
        -x 189 -y 32 -width 151 -height 19 -anchor nw -bordermode ignore
    place $base.fra19.lab17 \
        -x 20 -y 33 -width 56 -height 17 -anchor nw -bordermode ignore
    place $base.fra19.che17 \
        -x 25 -y 12 -width 101 -height 22 -anchor nw -bordermode ignore
    place $base.fra19.che18 \
        -x 198 -y 12 -width 148 -height 22 -anchor nw -bordermode ignore
    place $base.lab19 \
        -x 15 -y 280 -width 106 -height 17 -anchor nw -bordermode ignore
}

proc vTclWindow.top18 {base {container 0}} {
    if {$base == ""} {
        set base .top18
    }
    if {[winfo exists $base] && (!$container)} {
        wm deiconify $base; return
    }

    global widget

    ####################
    # CREATING WIDGETS
    ####################
    if {!$container} {
    toplevel $base -class Toplevel
    wm withdraw .top18
    wm focusmodel $base passive
    wm geometry $base 340x204+100+100; update
```

```
    wm maxsize $base 1024 768
    wm minsize $base 106 2
    wm overrideredirect $base 0
    wm resizable $base 0 0
    wm title $base "Geometric correction"
    }
    button $base.but18 \
        \
        -command {if { $bLog == 1 } { \
    set sLog "-l" \
} else { \
    set sLog "" \
}
if { $bPPM == 1 } { \
    set sPPM "-p" \
} else { \
    set sPPM "" \
}
if { $bPolygon == 1 } { \
    set sPolygon "-y";
    append sPolygon $sPolygonFile \
} else { \
    set sPolygon "" \
}
catch { exec CorGeo.exe -w$iWidth -h$iHeight -r$imgRImage -c$imgCImage \
    -t$imgOImage $sPolygon $sLog $sPPM } sStdOut
if { [string last done $sStdOut [string length $sStdOut]] > 0 } { \
    .top18.fra19.tex18 delete 0.0 [.top18.fra19.tex18 index end];
    .top18.fra19.tex18 insert 0.0 $sStdOut;
    if { $bLog == 1 } { \
        .top18.but19 configure -state normal \
    } else { \
        .top18.but19 configure -state disable \
    };
    if { $bPPM == 1 } { \
        Window show .top23
        focus .top23
        image create photo imgResult -file $imgOImage
        .top23.lab23 configure -image imgResult \
    } \
} else { \
    .top18.fra19.tex18 delete 0.0 [.top18.fra19.tex18 index end];
    .top18.fra19.tex18 insert 0.0 "The process has failed" \
}} \
        -pady 0 -text Run -underline 0
    button $base.but19 \
        \
        -command {Window show .top22
focus .top22 wm title .top22 "Log file" set fLog [open DetCam.log r] set sLog
[read $fLog] close $fLog .top22.tex22 delete 0.0 [.top22.tex22 index end]
.top22.tex22 insert 0.0 $sLog} \
        -pady 0 -state disabled -text {View log file} -underline 0
    button $base.but20 \
        -command {Window hide .top18} -pady 0 -text Cancel -underline 0
    frame $base.fra18 \
        -borderwidth 2 -cursor arrow -height 75 -relief groove -width 125
    checkbutton $base.fra18.che18 \
        \
        -command {if { $bPolygon == 0 } { \
    .top18.fra18.ent18 configure -state disable;
    .top18.fra18.but18 configure -state disable \
} else { \
```

```
    .top18.fra18.ent18 configure -state normal;
    .top18.fra18.but18 configure -state normal \
}} \
        -text {Polygon file} -underline 0 -variable bPolygon
    entry $base.fra18.ent18 \
        -state disabled -textvariable sPolygonFile
    button $base.fra18.but18 \
        \
        -command {set types { {{Polygon files} {.POL}} {{All files} *} }
set sPolygonFile [tk_getOpenFile -filetypes $types]} \
        -pady 0 -state disabled -text <<
    frame $base.fra19 \
        -borderwidth 2 -height 75 -relief groove -width 125
    text $base.fra19.tex18 \
        -yscrollcommand "$base.fra19.scr18 set"
    scrollbar $base.fra19.scr18 \
        -command "$base.fra19.tex18 yview"
    label $base.lab18 \
        -borderwidth 1 -text {Additional arguments}
    ####################
    # SETTING GEOMETRY
    ####################
    place $base.but18 \
        -x 10 -y 168 -width 80 -height 25 -anchor nw -bordermode ignore
    place $base.but19 \
        -x 128 -y 168 -width 80 -height 25 -anchor nw -bordermode ignore
    place $base.but20 \
        -x 248 -y 168 -width 80 -height 25 -anchor nw -bordermode ignore
    place $base.fra18 \
        -x 9 -y 86 -width 320 -height 70 -anchor nw -bordermode ignore
    place $base.fra18.che18 \
        -x 6 -y 9 -width 87 -height 27 -anchor nw -bordermode ignore
    place $base.fra18.ent18 \
        -x 13 -y 39 -width 251 -height 19 -anchor nw -bordermode ignore
    place $base.fra18.but18 \
        -x 276 -y 40 -width 30 -height 18 -anchor nw -bordermode ignore
    place $base.fra19 \
        -x 9 -y 12 -width 320 -height 60 -anchor nw -bordermode ignore
    place $base.fra19.tex18 \
        -x 8 -y 9 -width 286 -height 43 -anchor nw -bordermode ignore
    place $base.fra19.scr18 \
        -x 295 -y 8 -width 16 -height 44 -anchor nw -bordermode ignore
    place $base.lab18 \
        -x 15 -y 77 -width 106 -height 17 -anchor nw -bordermode ignore
}

proc vTclWindow.top19 {base {container 0}} {
    if {$base == ""} {
        set base .top19
    }
    if {[winfo exists $base] && (!$container)} {
        wm deiconify $base; return
    }

    global widget

    ####################
    # CREATING WIDGETS
    ####################
    if {!$container} {
    toplevel $base -class Toplevel \
        -cursor arrow
```

```
    wm withdraw .top19
    wm focusmodel $base passive
    wm geometry $base 340x120+100+100; update
    wm maxsize $base 1024 768
    wm minsize $base 106 2
    wm overrideredirect $base 0
    wm resizable $base 0 0
    wm title $base "Radiometric correction"
    }
    button $base.but19 \
        \
        -command {if { $bLog == 1 } { \
    set sLog "-l" \
} else { \
    set sLog "" \
}
if { $bPPM == 1 } { \
    set sPPM "-p" \
} else { \
    set sPPM "" \
}
catch { exec CorRad.exe -w$iWidth -h$iHeight -r$imgRImage -c$imgCImage \
    -t$imgOImage $sLog $sPPM } sStdOut
if { [string last done $sStdOut [string length $sStdOut]] > 0 } { \
    .top19.fra19.tex19 delete 0.0 [.top19.fra19.tex19 index end];
    .top19.fra19.tex19 insert 0.0 $sStdOut;
    if { $bLog == 1 } { \
        .top19.but20 configure -state normal \
    } else { \
        .top19.but20 configure -state disable \
    };
    if { $bPPM == 1 } { \
        Window show .top23
        focus .top23
        image create photo imgResult -file $imgOImage
        .top23.lab23 configure -image imgResult \
    } \
} else { \
    .top19.fra19.tex19 delete 0.0 [.top19.fra19.tex19 index end];
    .top19.fra19.tex19 insert 0.0 "The process has failed" \
}} \
        -pady 0 -text Run -underline 0
    button $base.but20 \
        \
        -command {Window show .top22
focus .top22 wm title .top22 "Log file" set fLog [open DetCam.log r] set sLog
[read $fLog] close $fLog .top22.tex22 delete 0.0 [.top22.tex22 index end]
.top22.tex22 insert 0.0 $sLog} \
        -pady 0 -state disabled -text {View log file} -underline 0
    button $base.but21 \
        -command {Window hide .top19} -pady 0 -text Cancel -underline 0
    frame $base.fra19 \
        -borderwidth 2 -height 75 -relief groove -width 125
    text $base.fra19.tex19 \
        -yscrollcommand "$base.fra19.scr19 set"
    scrollbar $base.fra19.scr19 \
        -command "$base.fra19.tex19 yview"
    ####################
    # SETTING GEOMETRY
    ####################
    place $base.but19 \
        -x 10 -y 83 -width 80 -height 25 -anchor nw -bordermode ignore
```

```
    place $base.but20 \
        -x 128 -y 83 -width 80 -height 25 -anchor nw -bordermode ignore
    place $base.but21 \
        -x 248 -y 83 -width 80 -height 25 -anchor nw -bordermode ignore
    place $base.fra19 \
        -x 9 -y 12 -width 320 -height 60 -anchor nw -bordermode ignore
    place $base.fra19.tex19 \
        -x 8 -y 9 -width 286 -height 43 -anchor nw -bordermode ignore
    place $base.fra19.scr19 \
        -x 295 -y 8 -width 16 -height 44 -anchor nw -bordermode ignore
}

proc vTclWindow.top20 {base {container 0}} {
    if {$base == ""} {
        set base .top20
    }
    if {[winfo exists $base] && (!$container)} {
        wm deiconify $base; return
    }

    global widget

    ###################
    # CREATING WIDGETS
    ###################
    if {!$container} {
    toplevel $base -class Toplevel
    wm withdraw .top20
    wm focusmodel $base passive
    wm geometry $base 340x200+100+100; update
    wm maxsize $base 1024 768
    wm minsize $base 106 2
    wm overrideredirect $base 0
    wm resizable $base 0 0
    wm title $base "Images difference"
    }
    button $base.but20 \
        \
        -command {if { $iMode == 0 } { \
    set sMode "-a" \
} else { \
    set sMode "-o";
    append sMode $iOffset \
}
if { $bLog == 1 } { \
    set sLog "-l" \
} else { \
    set sLog "" \
}
if { $bPPM == 1 } { \
    set sPPM "-p" \
} else { \
    set sPPM "" \
}
catch { exec DifIma.exe $sMode -w$iWidth -h$iHeight -r$imgRImage \
    -c$imgCImage -t$imgOImage $sLog $sPPM } sStdOut
if { [string last done $sStdOut [string length $sStdOut]] > 0 } { \
    .top20.fra21.tex20 delete 0.0 [.top20.fra21.tex20 index end];
    .top20.fra21.tex20 insert 0.0 $sStdOut;
    if { $bLog == 1 } { \
        .top20.but21 configure -state normal \
    } else { \
```

```
        .top20.but21 configure -state disable \
    };
    if { $bPPM == 1 } { \
        Window show .top23
        focus .top23
        image create photo imgResult -file $imgOImage
        .top23.lab23 configure -image imgResult \
    } \
} else { \
    .top20.fra21.tex20 delete 0.0 [.top20.fra21.tex20 index end];
    .top20.fra21.tex20 insert 0.0 "The process has failed" \
}} \
        -pady 0 -text Run -underline 0
    button $base.but21 \
        \
        -command {Window show .top22
focus .top22 wm title .top22 "Log file" set fLog [open DetCam.log r] set sLog
[read $fLog] close $fLog .top22.tex22 delete 0.0 [.top22.tex22 index end]
.top22.tex22 insert 0.0 $sLog} \
        -pady 0 -state disabled -text {View log file} -underline 0
    button $base.but22 \
        -command {Window hide .top20} -pady 0 -text Cancel -underline 0
    frame $base.fra20 \
        -borderwidth 2 -height 75 -relief groove -width 125
    radiobutton $base.fra20.rad20 \
        \
        -command {.top20.fra20.ent20 configure -state disable
.top20.fra20.lab20 configure -state disable} \
        -text {DNout = abs(DNref - DNcor)} -value 0 -variable iMode
    radiobutton $base.fra20.rad21 \
        \
        -command {.top20.fra20.ent20 configure -state normal
.top20.fra20.lab20 configure -state normal} \
        -text {DNout = DNref - DNcor + Offset} -value 1 -variable iMode
    entry $base.fra20.ent20 \
        -justify right -state disabled -textvariable iOffset
    label $base.fra20.lab20 \
        -borderwidth 1 -state disabled -text Offset
    frame $base.fra21 \
        -borderwidth 2 -height 75 -relief groove -width 125
    text $base.fra21.tex20 \
        -yscrollcommand "$base.fra21.scr20 set"
    scrollbar $base.fra21.scr20 \
        -command "$base.fra21.tex20 yview"
    label $base.lab20 \
        -borderwidth 1 -text {Additional arguments}
    ####################
    # SETTING GEOMETRY
    ####################
    place $base.but20 \
        -x 10 -y 164 -width 80 -height 25 -anchor nw -bordermode ignore
    place $base.but21 \
        -x 128 -y 164 -width 80 -height 25 -anchor nw -bordermode ignore
    place $base.but22 \
        -x 248 -y 164 -width 80 -height 25 -anchor nw -bordermode ignore
    place $base.fra20 \
        -x 9 -y 86 -width 320 -height 65 -anchor nw -bordermode ignore
    place $base.fra20.rad20 \
        -x 5 -y 6 -width 167 -height 27 -anchor nw -bordermode ignore
    place $base.fra20.rad21 \
        -x 5 -y 29 -width 184 -height 27 -anchor nw -bordermode ignore
    place $base.fra20.ent20 \
```

```
        -x 215 -y 34 -width 91 -height 19 -anchor nw -bordermode ignore
    place $base.fra20.lab20 \
        -x 212 -y 15 -width 37 -height 17 -anchor nw -bordermode ignore
    place $base.fra21 \
        -x 9 -y 12 -width 320 -height 60 -anchor nw -bordermode ignore
    place $base.fra21.tex20 \
        -x 8 -y 9 -width 286 -height 43 -anchor nw -bordermode ignore
    place $base.fra21.scr20 \
        -x 295 -y 8 -width 16 -height 44 -anchor nw -bordermode ignore
    place $base.lab20 \
        -x 15 -y 77 -width 106 -height 17 -anchor nw -bordermode ignore
}

proc vTclWindow.top21 {base {container 0}} {
    if {$base == ""} {
        set base .top21
    }
    if {[winfo exists $base] && (!$container)} {
        wm deiconify $base; return
    }

    global widget

    ###################
    # CREATING WIDGETS
    ###################
    if {!$container} {
    toplevel $base -class Toplevel
    wm withdraw .top21
    wm focusmodel $base passive
    wm geometry $base 340x280+100+100; update
    wm maxsize $base 1024 768
    wm minsize $base 106 2
    wm overrideredirect $base 0
    wm resizable $base 0 0
    wm title $base "Changes detection"
    }
    button $base.but21 \
        \
        -command {if { $iMode == 0 } { \
    set sMode "-a" \
} else { \
    set sMode "-o";
    append sMode $iOffset \
}
if { $bLog == 1 } { \
    set sLog "-l" \
} else { \
    set sLog "" \
}
if { $bImages == 0 } { \
    set sImages "-i" \
} else { \
    set sImages ""\
}
if { $bPPM == 1 } { \
    set sPPM "-p" \
} else { \
    set sPPM "" \
}
if { $bPolygon == 1 } { \
    set sPolygon "-y";
```

```
    append sPolygon $sPolygonFile \
} else { \
    set sPolygon "" \
} catch { exec DetCam.exe $sMode -w$iWidth -h$iHeight -r$imgRImage \
    -c$imgCImage -t$imgOImage $sPolygon $sLog $sImages $sPPM } sStdOut
if { [string last done $sStdOut [string length $sStdOut]] > 0 } { \
    .top21.fra22.tex21 delete 0.0 [.top21.fra22.tex21 index end];
    .top21.fra22.tex21 insert 0.0 $sStdOut;
    if { $bLog == 1 } { \
        .top21.but22 configure -state normal \
    } else { \
        .top21.but22 configure -state disable \
    };
    if { $bPPM == 1 } { \
        Window show .top23
        focus .top23
        image create photo imgResult -file $imgOImage
        .top23.lab23 configure -image imgResult \
    } \
} else { \
    .top21.fra22.tex21 delete 0.0 [.top21.fra22.tex21 index end];
    .top21.fra22.tex21 insert 0.0 "The process has failed" \
}} \
        -pady 0 -text Run -underline 0
    button $base.but22 \
        \
        -command {Window show .top22
focus .top22 wm title .top22 "Log file" set fLog [open DetCam.log r] set sLog
[read $fLog] close $fLog .top22.tex22 delete 0.0 [.top22.tex22 index end]
.top22.tex22 insert 0.0 $sLog} \
        -pady 0 -text {View log file} -underline 0
    button $base.but23 \
        -command {Window hide .top21} -pady 0 -text Cancel -underline 0
    frame $base.fra21 \
        -borderwidth 2 -height 75 -relief groove -width 125
    radiobutton $base.fra21.rad21 \
        \
        -command {.top21.fra21.ent21 configure -state disable
.top21.fra21.lab21 configure -state disable} \
        -text {DNout = abs(DNref - DNcor)} -value 0 -variable iMode
    radiobutton $base.fra21.rad22 \
        \
        -command {.top21.fra21.ent21 configure -state normal
.top21.fra21.lab21 configure -state normal} \
        -text {DNout = DNref - DNcor + Offset} -value 1 -variable iMode
    entry $base.fra21.ent21 \
        -justify right -state disabled -textvariable iOffset
    label $base.fra21.lab21 \
        -borderwidth 1 -state disabled -text Offset
    checkbutton $base.fra21.che21 \
        \
        -command {if { $bPolygon == 0 } { \
    .top21.fra21.ent22 configure -state disable;
    .top21.fra21.but21 configure -state disable \
} else { \
    .top21.fra21.ent22 configure -state normal;
    .top21.fra21.but21 configure -state normal \
}} \
        -text {Polygon file} -underline 0 -variable bPolygon
    entry $base.fra21.ent22 \
        -textvariable sPolygonFile
    button $base.fra21.but21 \
```

```
        \
        -command {set types { {{Polygon files} {.POL}} {{All files} *} }
set sPolygonFile [tk_getOpenFile -filetypes $types]} \
        -pady 0 -text <<
    checkbutton $base.fra21.che22 \
        -justify left -text {Remove intermediate images} -underline 0 \
        -variable bImages
    frame $base.fra22 \
        -borderwidth 2 -height 75 -relief groove -width 125
    text $base.fra22.tex21 \
        -yscrollcommand "$base.fra22.scr21 set"
    scrollbar $base.fra22.scr21 \
        -command "$base.fra22.tex21 yview"
    label $base.lab21 \
        -borderwidth 1 -text {Additional arguments}
    ####################
    # SETTING GEOMETRY
    ####################
    place $base.but21 \
        -x 10 -y 244 -width 80 -height 25 -anchor nw -bordermode ignore
    place $base.but22 \
        -x 128 -y 244 -width 80 -height 25 -anchor nw -bordermode ignore
    place $base.but23 \
        -x 248 -y 244 -width 80 -height 25 -anchor nw -bordermode ignore
    place $base.fra21 \
        -x 9 -y 86 -width 320 -height 145 -anchor nw -bordermode ignore
    place $base.fra21.rad21 \
        -x 5 -y 6 -width 167 -height 27 -anchor nw -bordermode ignore
    place $base.fra21.rad22 \
        -x 5 -y 29 -width 184 -height 27 -anchor nw -bordermode ignore
    place $base.fra21.ent21 \
        -x 215 -y 34 -width 91 -height 19 -anchor nw -bordermode ignore
    place $base.fra21.lab21 \
        -x 212 -y 15 -width 37 -height 17 -anchor nw -bordermode ignore
    place $base.fra21.che21 \
        -x 5 -y 56 -width 87 -height 27 -anchor nw -bordermode ignore
    place $base.fra21.ent22 \
        -x 12 -y 86 -width 251 -height 19 -anchor nw -bordermode ignore
    place $base.fra21.but21 \
        -x 275 -y 86 -width 30 -height 18 -anchor nw -bordermode ignore
    place $base.fra21.che22 \
        -x 8 -y 112 -width 163 -height 27 -anchor nw -bordermode ignore
    place $base.fra22 \
        -x 9 -y 12 -width 320 -height 60 -anchor nw -bordermode ignore
    place $base.fra22.tex21 \
        -x 8 -y 9 -width 286 -height 43 -anchor nw -bordermode ignore
    place $base.fra22.scr21 \
        -x 295 -y 8 -width 16 -height 44 -anchor nw -bordermode ignore
    place $base.lab21 \
        -x 15 -y 77 -width 106 -height 17 -anchor nw -bordermode ignore
}

proc vTclWindow.top22 {base {container 0}} {
    if {$base == ""} {
        set base .top22
    }
    if {[winfo exists $base] && (!$container)} {
        wm deiconify $base; return
    }

    global widget
```

```
    ####################
    # CREATING WIDGETS
    ####################
    if {!$container} {
    toplevel $base -class Toplevel \
        -cursor arrow
    wm withdraw .top22
    wm focusmodel $base passive
    wm geometry $base 512x256+100+100; update
    wm maxsize $base 1024 768
    wm minsize $base 106 2
    wm overrideredirect $base 0
    wm resizable $base 1 1
    wm title $base "Log file"
    }
    scrollbar $base.scr22 \
        -command "$base.tex22 yview"
    text $base.tex22 \
        -height 30 -wrap word -yscrollcommand "$base.scr22 set"
    ####################
    # SETTING GEOMETRY
    ####################
    pack $base.scr22 \
        -in $base -anchor center -expand 0 -fill y -side right
    pack $base.tex22 \
        -in $base -anchor center -expand 1 -fill both -side top
}

proc vTclWindow.top23 {base {container 0}} {
    if {$base == ""} {
        set base .top23
    }
    if {[winfo exists $base] && (!$container)} {
        wm deiconify $base; return
    }

    global widget

    ####################
    # CREATING WIDGETS
    ####################
    if {!$container} {
    toplevel $base -class Toplevel
    wm withdraw .top23
    wm focusmodel $base passive
    wm geometry $base 256x256+100+100; update
    wm maxsize $base 1024 768
    wm minsize $base 106 2
    wm overrideredirect $base 0
    wm resizable $base 1 1
    wm title $base "Result image (PPM)"
    }
    label $base.lab23 \
        -borderwidth 1
    ####################
    # SETTING GEOMETRY
    ####################
    pack $base.lab23 \
        -in $base -anchor center -expand 1 -fill both -side top
}

Window show .
```

```
Window show .top17
Window show .top18
Window show .top19
Window show .top20
Window show .top21
Window show .top22
Window show .top23

main $argc $argv
```

Apéndice B

Manual

B.1 Introducción

Como ya comentamos en el capítulo 1.2, el objetivo de este proyecto es el desarrollo de una aplicación que automatice los procesos necesarios para el estudio de series multi-temporales de imágenes con objeto de detectar cambios urbanos.

Los datos proporcionados por esta aplicación deberán ser utilizados por un GIS, concretamente, el GIS de la gerencia de Urbanismo del Iltmo. Ayuntamiento de Málaga, para determinar la legalidad o ilegalidad de los cambios urbanos detectados.

El objetivo inicial ha sido alcanzado y como resultado, disponemos de un conjunto de módulos que realizan el proceso detección de cambios (o alguna de sus etapas) (ver figura B.1), atendiendo a características específicas de las regiones urbanas (variabilidad espacial, etc.).

> **NOTA:** Hemos creído conveniente desglosar todas y cada una de las etapas del proceso de detección para incorporarlas individualmente a GIS comerciales (por ejemplo, GRASS).

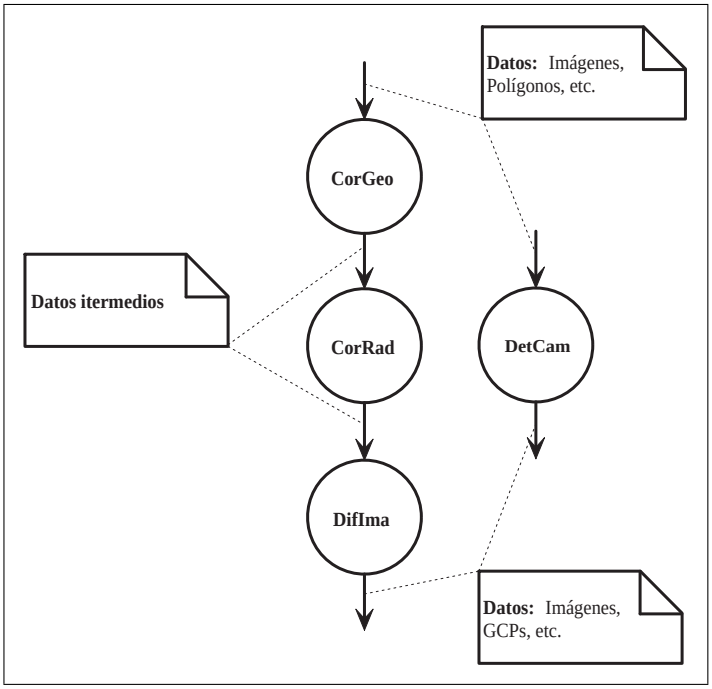


Figura B.1: Módulos individuales *vs.* Módulo global.

B.1.1 Modos de operación

Según lo expuesto hasta el momento, podemos distinguir dos modos de operación:

1. Operación en consola (línea de comandos) (ver figura B.2)

2. Operación en interfaz gráfico (ver figura B.3)

La diferencia entre ambos modos de operación radica en la facilidad de manejo, y en la posibilidad de estudiar, fácilmente, los resultados obtenidos: imágenes, GCPs, etc. Sin embargo, la posibilidad de ejecutar los módulos desde la línea de comandos facilita el procesamiento por lotes (archivos *batch*) y evita la necesidad de disponer de un entorno gráfico para ejecutar el interfaz.

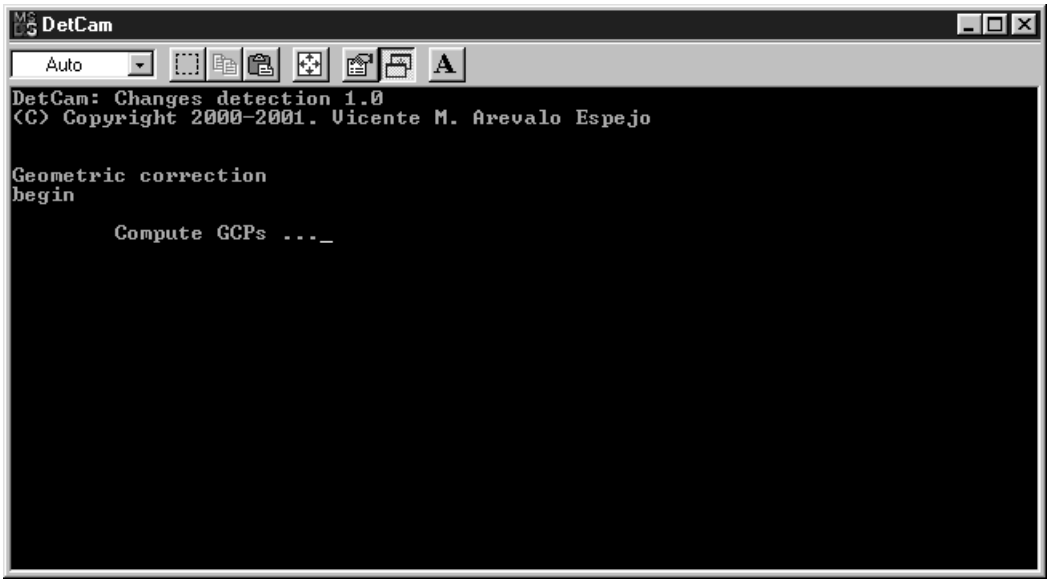


Figura B.2: Consola MS-DOS.

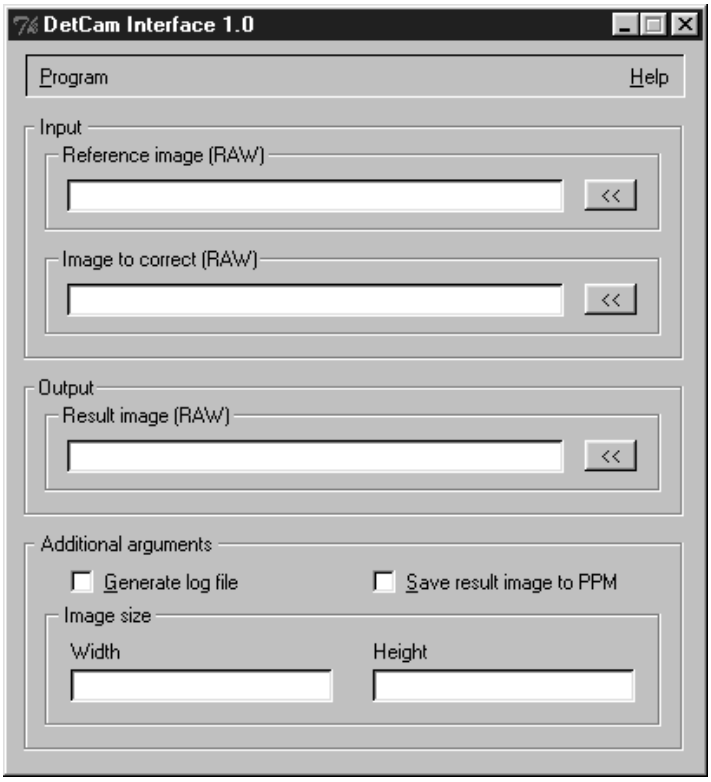


Figura B.3: Interfaz gráfico TCL/TK.

B.2 Requisitos mínimos

Los requisitos mínimos exigibles para ejecutar con garantías de éxito los módulos y el interfaz gráfico son los siguientes:

> **NOTA:** A la hora de estimar los requisitos mínimos se han considerado tanto recursos software como hardware, atendiendo a las características de los módulos y del tipo de procesamiento a realizar.

B.2.1 Software

Los recursos software que se requieren para poder ejecutar los módulos y el interfaz gráfico son los siguientes:

Módulos

Los requisitos mínimos exigibles son:

1. Sistema operativo Windows (Consola MS-DOS)

2. Sistema operativo MS-DOS

Interfaz gráfico

Los requisitos mínimos exigibles son:

1. Sistema operativo Windows

2. TCL/TK 8.3 o superior

B.2.2 Hardware

Los recursos hardware que se requieren para poder ejecutar los módulos y el interfaz gráfico son los siguientes:

Módulos e interfaz gráfico

Los requisitos mínimos exigibles son:

1. PC compatible

   - `Descripción`
     PC compatible con procesador Intel PII (o equivalente)- 250MHz, 128MB de memoria estándar y 16MB de memoria de vídeo.

Sin embargo, los requisitos recomendados son:

1. PC compatible

   - `Descripción`
     PC compatible con procesador Intel PIII (o equivalente)- 600MHz, 256MB de memoria estándar y 32MB de memoria de vídeo.

Recordar que el tamaño de las imágenes de satélite utilizadas a lo largo de este proyecto rondan aproximadamente los 20MB de media. Por lo tanto, es posible que durante el proceso de detección de cambios la cantidad de memoria estándar demandada se dispare.

B.3 Instalación

Para instalar la aplicación (módulos, interfaz gráfico y ejemplos) en un *PC compatible* bajo *Windows 98* es necesario realizar los siguientes pasos:

1. Insertar el CD en el lector de CD-ROM
2. Ejecutar `Setup.exe` desde el *Panel de control* o el *Explorador de Windows*
3. Seleccionar el directorio de trabajo
4. Seleccionar el tipo de instalación
5. Seleccionar la carpeta de programas

B.3.1 Modo de operación

Los pasos necesarios para llevar a cabo el proceso de instalación son los siguientes:

> **NOTA:** El proceso de instalación se realizará desde el *Panel de control*. El proceso de instalación desde el *Explorador de Windows* es análogo.

Escritorio

Situarse en el *Escritorio de Windows 98* (ver figura B.4). Las operaciones a realizar son las siguientes:

1. Hacer *click* en el botón `Inicio`
2. Hacer *click* en el icono `Configuración`
3. Hacer *click* en el icono `Panel de control`

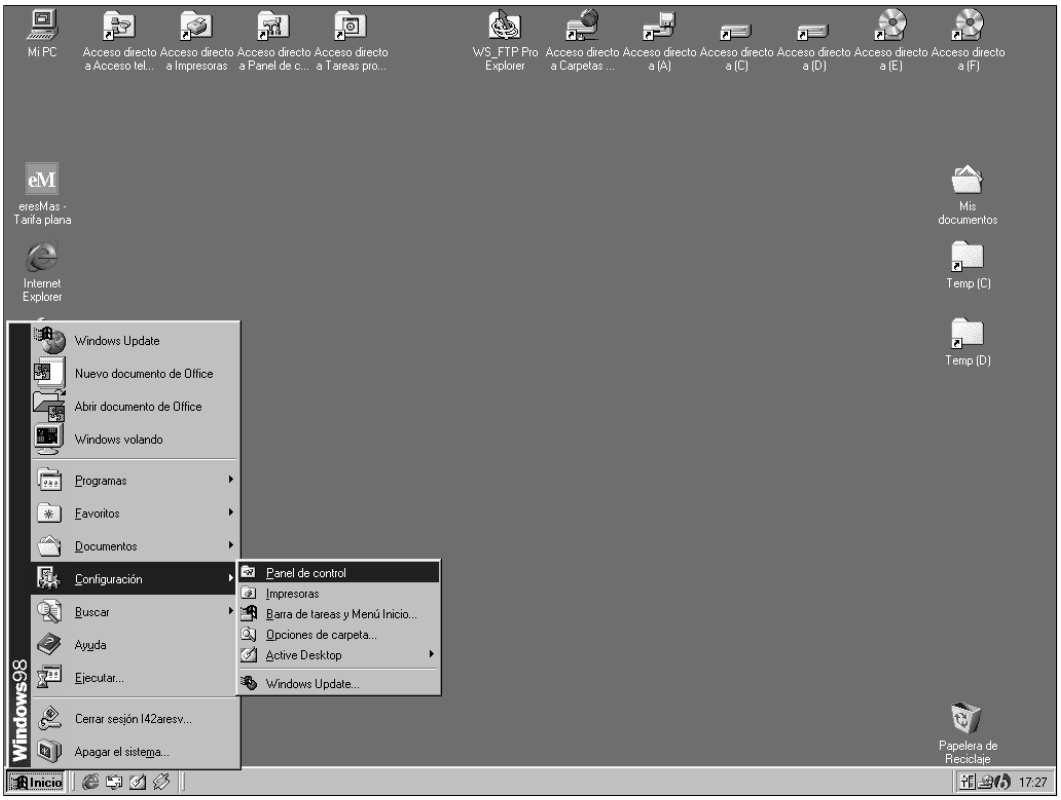


Figura B.4: Escritorio de Windows 98.

Tras realizar estas operaciones se visualiza la ventana *Panel de control*.

Panel de control

Situarse en la ventana *Panel de control* (ver figura B.5). Las operaciones a realizar son las siguientes:

1. Hacer *doble click* en el icono `Agregar o quitar programas`

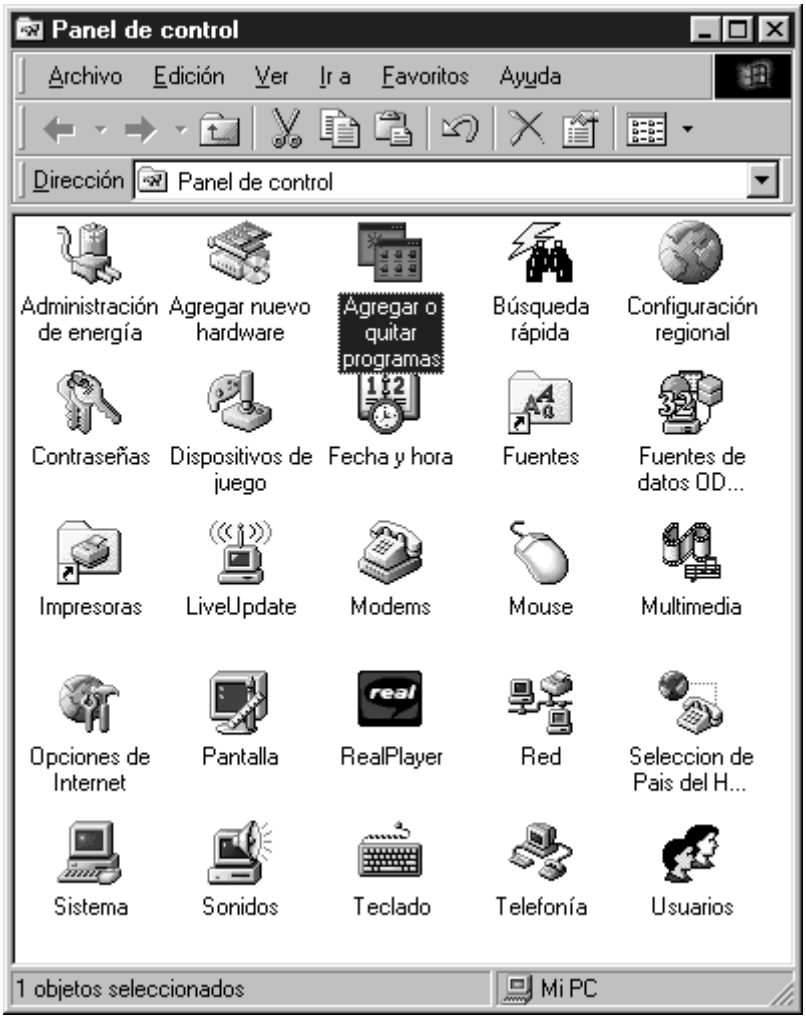


Figura B.5: Panel de control.

Tras realizar estas operaciones se visualiza la ventana *Propiedades de Agregar o quitar programas*.

Propiedades de Agregar o quitar programas

Situarse en la ventana *Propiedades de Agregar o quitar programas* (ver figura B.6). Las operaciones a realizar son las siguientes:

1. Pulsar return (el botón por defecto es `Instalar`) para continuar con el proceso de instalación

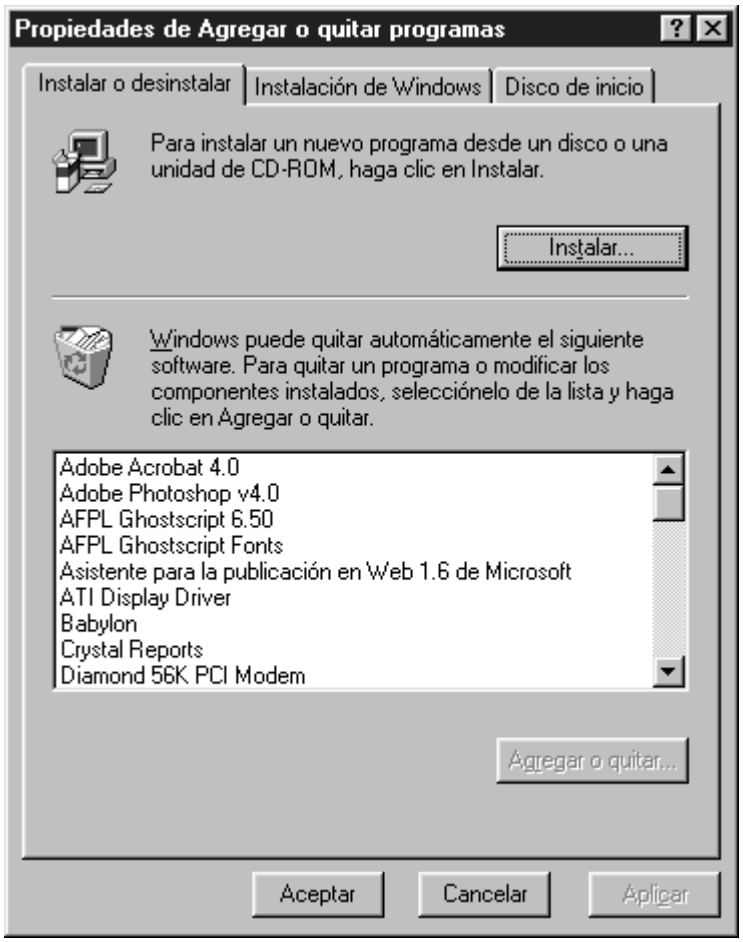


Figura B.6: Propiedades de Agregar o quitar programas.

Tras realizar estas operaciones se visualiza la ventana *Instalar programa desde disco o CD-ROM*.

Instalar programa desde disco o CD-ROM

Situarse en la ventana *Instalar programa desde disco o CD-ROM* (ver figura B.7). Las operaciones a realizar son las siguientes:

1. Pulsar return (el botón por defecto es `Siguiente`) para continuar con el proceso de instalación

Figura B.7: Instalar programa desde disco o CD-ROM.

Tras realizar estas operaciones se visualiza la ventana *Ejecutar el programa de instalación.*

Ejecutar el programa de instalación

Situarse en la ventana *Ejecutar el programa de instalación* (ver figura B.8). Las operaciones a realizar son las siguientes:

1. Pulsar return (el botón por defecto es `Finalizar`) para continuar con el proceso de instalación

Figura B.8: Ejecutar el programa de instalación.

Tras realizar estas operaciones se visualiza la ventana *Welcome*.

Welcome

Situarse en la ventana *Welcome* (ver figura B.9). Las operaciones a realizar son las siguientes:

1. Pulsar return (el botón por defecto es `Next`) para continuar con el proceso de instalación

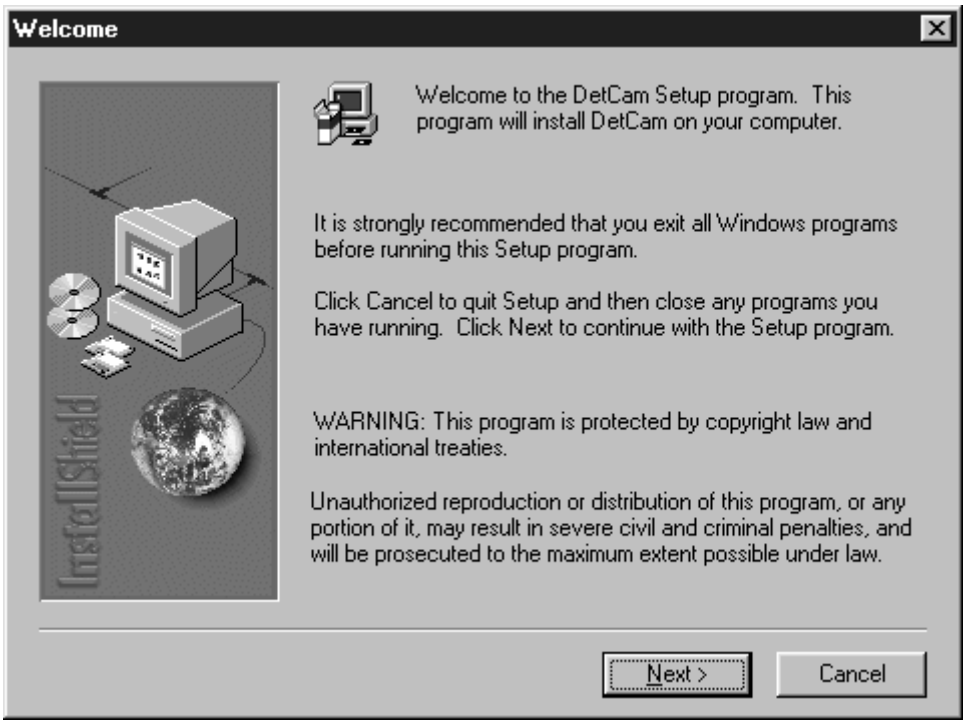


Figura B.9: Welcome.

Tras realizar estas operaciones se visualiza la ventana *Choose destination location*.

Choose destination location

Situarse en la ventana *Choose destination location* (ver figura B.10).

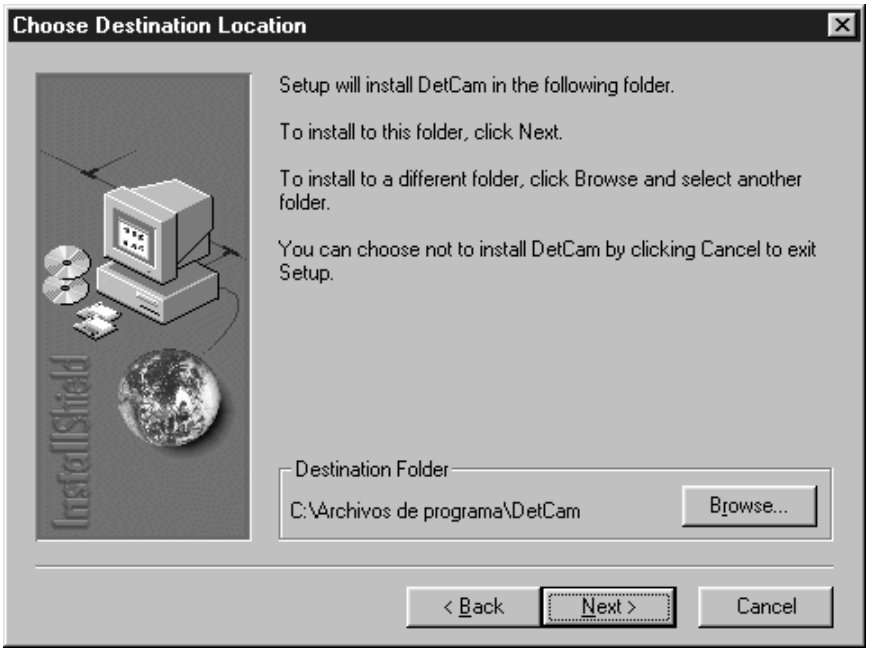


Figura B.10: Choose destination location.

Opción 1

Se acepta el directorio destino establecido por defecto. Las operaciones a realizar son las siguientes:

1. Pulsar return (el botón por defecto es `Next`) para continuar con el proceso de instalación

Tras realizar estas operaciones se visualiza la ventana *Setup type*.

Opción 2

No se acepta el directorio destino establecido por defecto. Las operaciones a realizar son las siguientes:

1. Hacer *click* en el botón `Browse`

Tras realizar estas operaciones se visualiza la ventana *Choose folder*.

Choose folder

Situarse en la ventana *Choose folder* (ver figura B.11). Las operaciones a realizar son las siguientes:

1. Seleccionar el directorio destino
2. Pulsar `return` (el botón por defecto es `OK`) para continuar con el proceso de instalación

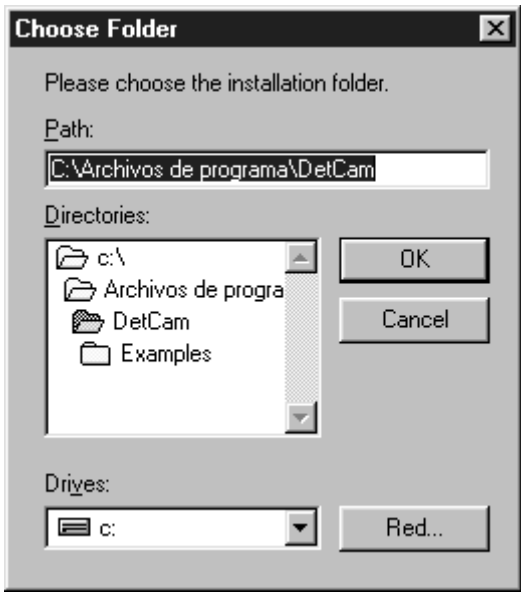


Figura B.11: Choose folder.

Tras realizar estas operaciones se visualiza la ventana *Setup type*.

> **NOTA:** Sólo si en la ventana anterior ha sido pulsado el botón `Browse`.

Setup type

Situarse en la ventana *Setup type* (ver figura B.12).

Figura B.12: Setup type.

Opción 1

Se selecciona el tipo de instalación `Typical` o `Compact`. Las operaciones a realizar son las siguientes:

1. Pulsar return (el botón por defecto es `Next`) para continuar con el proceso de instalación

Tras realizar estas operaciones se visualiza la ventana *Select program folder*.

Opción 2

Se selecciona el tipo de instalación `Custom`. Las operaciones a realizar son las siguientes:

1. Pulsar return (el botón por defecto es `Next`) para continuar con el proceso de instalación

Tras realizar estas operaciones se visualiza la ventana *Select components*.

Select components

Situarse en la ventana *Select components* (ver figura B.13). Las operaciones a realizar son las siguientes:

1. Seleccionar los componentes:
   (a) Program files (ejecutables)
   (b) Examples files (imágenes y polígonos de interés)
2. Pulsar `return` (el botón por defecto es `Next`) para continuar con el proceso de instalación

Figura B.13: Select components.

Tras realizar estas operaciones se visualiza la ventana *Select program folder*.

> **NOTA:** Sólo si en la ventana anterior ha sido seleccionada la opción `Custom`.

Select program folder

Situarse en la ventana *Select program folder* (ver figura B.14). Las operaciones a realizar son las siguientes:

1. Seleccionar la carpeta de programas
2. Pulsar return (el botón por defecto es `Next`) para finalizar el proceso de instalación

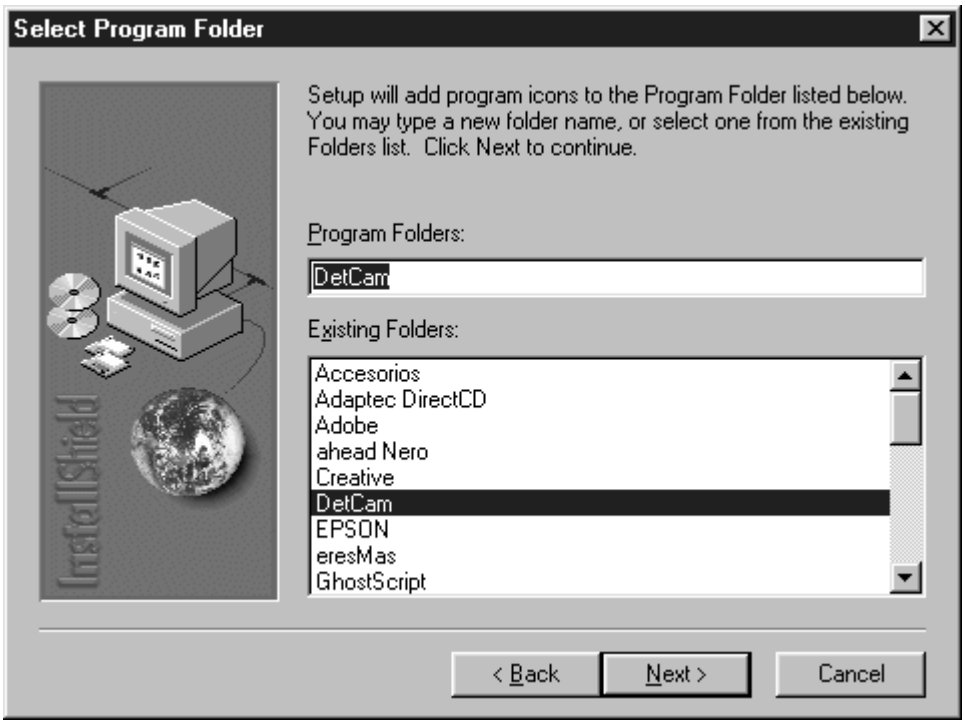


Figura B.14: Select program folder.

Tras realizar estas operaciones el proceso de instalación finaliza.

B.4 Descripción

En esta sección realizaremos una descripción pormenorizada del funcionamiento tanto de los módulos como del interfaz gráfico. Para ello se utilizarán varias imágenes de ejemplo que ilustrarán convenientemente los procesos a realizar.

Imágenes

A lo largo de esta sección vamos a utilizar dos series multi-temporales de imágenes para realizar los diferentes procesos:

Málaga

La imagen a corregir (Málaga'00) y la de referencia (Málaga'99) se encuentran desplazadas 15 píxeles en x y 9 píxeles en y, aproximadamente. Sus dimensiones son de 512×512 píxeles (2.56×2.56 km).

Figura B.15: Imagen de referencia.

Figura B.16: Imagen a corregir.

Benalmádena

La imagen a corregir (Benalmádena'99) y la de referencia (Benalmádena'00) se encuentran desplazadas 1 píxel en x y 2 píxeles en y, aproximadamente. Sus dimensiones son de 512×512 píxeles (2.56×2.56 km).

Figura B.17: Imagen de referencia.

Figura B.18: Imagen a corregir.

Polígonos

A lo largo de esta sección vamos a utilizar dos ficheros POI (Polygon Of Interest) para realizar los diferentes procesos.

Los ficheros que almacenan POIs tienen una estructura muy simple:

- La primera línea contiene el número de vértices del polígono más uno
- El resto de las líneas incluyen las coordenadas x e y de los vértices

Los polígonos utilizados son cerrados y como tales, la última línea, correspondiente al último vértice, coincide con la segunda, correspondiente al primer vértice del polígono.

Málaga

El polígono de interés utilizado en aquellos procesos en los que se empleen las imágenes de la serie multi-temporal de Málaga (figuras B.15 y B.16) es el siguiente:

```
5
92  198
464 325
486 207
65  314
92  198
```

Benalmádena

El polígono de interés utilizado en aquellos procesos en los que se empleen las imágenes de la serie multi-temporal de Benalmádena (figuras B.17 y B.18) es el siguiente:

```
7
86  66
241 63
463 314
442 446
308 443
51  186
86  66
```

B.4.1 Módulos

En este apartado realizaremos una descripción pormenorizada del funcionamiento de los distintos módulos.

- `CorGeo`: Corrección geométrica
- `CorRad`: Corrección radiométrica
- `DifIma`: Diferencia de imágenes
- `DetCam`: Detección de cambios (corrección geométrica y radiométrica, y diferencia de imágenes).

  NOTA: Como ya comentamos en la sección 6.2 una de las restricciones obligadas por la técnica de corrección geométrica utilizada era la utilización de imágenes del mismo tamaño y con dimensiones (alto y ancho) múltiplos de la dimensión del *tile* elegido (en nuestro caso 128×128 píxeles).

Módulo CorGeo

El módulo `CorGeo` realiza la corrección geométrica de una imagen (imagen a corregir) con respecto a otra (imagen de referencia).

Modo de operación

Ejemplo 1: Ayuda

1. Ejecutar `CorGeo` con la opción `-?`

```
D:\>CorGeo -?
CorGeo: Geometric correction 1.0
(C) Copyright 2000-2001. Vicente M. Arevalo Espejo

Usage: CorGeo -? -wWidth -hHeight -rRefImg -cCorImg -tOutImg [-l] [-p] [-yPolygon]
        -?:     Display this information
        -wWidth:        Image width (Width % 128 = 0)
        -hHeight:       Image height (Height % 128 = 0)
        -rRefImg:       Reference image
        -cCorImg:       Image to correct
        -tOutImg:       Result image
        -l:     Log file
        -p:     Save result image to PPM format
        -yPolygon:      Polygon of interest
```

Ejemplo 2: Corrección geométrica

1. Ejecutar `CorGeo` con las opciones:

   - `-w512`
   - `-h512`
   - `-ri_99(mala)512.raw`
   - `-ci_00(mala)512.raw`
   - `-ti_00(mala)512g.raw`
   - `-l`
   - `-p` (Si deseamos que `CorGeo` guarde la imagen resultado en formato PPM)
   - `-y` (Si deseamos que `CorGeo` realice la corrección geométrica en una región de interés)

   NOTA: Si omitimos la opción `-yPolygon` la región de interés es toda la imagen.

```
D:\>CorGeo -w512 -h512 -ri_99(mala)512.raw -ci_00(mala)512.raw -ti_00(mala)512g.raw -l
CorGeo: Geometric correction 1.0
(C) Copyright 2000-2001. Vicente M. Arevalo Espejo

Geometric correction
begin

        Compute GCPs ..... done

        Remove bad GCPs ..... done
```

```
        Compute coefficients ...... done

        Correct image ...... done

end
```

El fichero *log* que genera `CorGeo` es el siguiente:

```
// Geometric correction
// ********************
// begin
[Geometric correction]
[Tile width]
128
[Tile height]
128
[X tile number]
4
[Y tile number]
4
[Tile number]
16
// KLT algorithm
[Number of features]
25
[Window width]
15
[Window height]
15
// -1: Bad tile; -2: Tile out of polygon
// Tile.y, Tile.x, R.x, R.y, C.x, C.y, CX, CY
[Tracked points]
0   0   80.000000   51.000000   64.911484   60.801441   15.088516   -9.801441
0   1   193.000000  62.000000   181.777557  46.557648   11.222443   15.442352
0   2   292.000000  55.000000   288.436596  49.631901   3.563404    5.368099
0   3   485.000000  45.000000   469.643219  52.944145   15.356781   -7.944145
1   0   -1.000000   -1.000000   -1.000000   -1.000000   0.000000    0.000000
1   1   221.000000  159.000000  205.408981  170.568642  15.591019   -11.568642
1   2   357.000000  165.000000  341.301376  173.089973  15.698624   -8.089973
1   3   477.000000  157.000000  461.327713  165.482704  15.672287   -8.482704
2   0   53.000000   286.000000  38.073761   297.015652  14.926239   -11.015652
2   1   -1.000000   -1.000000   -1.000000   -1.000000   0.000000    0.000000
2   2   -1.000000   -1.000000   -1.000000   -1.000000   0.000000    0.000000
2   3   444.000000  286.000000  428.203423  294.332935  15.796577   -8.332935
3   0   103.000000  408.000000  88.773430   416.773643  14.226570   -8.773643
3   1   190.000000  483.000000  193.149445  473.766220  -3.149445   9.233780
3   2   349.000000  426.000000  335.676857  434.625546  13.323143   -8.625546
3   3   -1.000000   -1.000000   -1.000000   -1.000000   0.000000    0.000000
[Number of tracked points]
12
// Tile.y, Tile.x, R.x, R.y, C.x, C.y, CX, CY
[Deleted points]
3   1   190.000000  483.000000  193.149445  473.766220  -3.149445   9.233780
0   1   193.000000  62.000000   181.777557  46.557648   11.222443   15.442352
0   2   292.000000  55.000000   288.436596  49.631901   3.563404    5.368099
1   1   221.000000  159.000000  205.408981  170.568642  15.591019   -11.568642
[End]
[Number of deleted points]
4
// x' = a0 + a1 * x + a2 * y + a3 * x * y
```

```
// y' = b0 + b1 * x + b2 * y + b3 * x * y
[Coefficients]
15.432620   1.001621    -0.002956   -0.000003
-10.509747  0.005384    1.001434    -0.000004
//  end
```

La imagen corregida geométricamente por `CorGeo` es la siguiente:

Figura B.19: Imagen corregida geométricamente.

Módulo CorRad

El módulo `CorRad` realiza la corrección radiométrica[1] de una imagen (imagen a corregir) con respecto a otra (imagen de referencia).

Modo de operación

Ejemplo 1: Ayuda

1. Ejecutar `CorRad` con la opción **-?**

```
D:\>CorRad -?
CorRad: Radiometric correction 1.0
(C) Copyright 2000-2001. Vicente M. Arevalo Espejo

Usage: CorRad -? -wWidth -hHeight -rRefImg -cCorImg -tOutImg [-l] [-p]
        -?:     Display this information
        -wWidth:        Image width (Width % 128 = 0)
        -hHeight:       Image height (Height % 128 = 0)
        -rRefImg:       Reference image
```

[1] Especificación del histograma.

```
-cCorImg:       Image to correct
-tOutImg:       Result image
-l:     Log file
-p:     Save result image to PPM format
```

Ejemplo 2: Corrección radiométrica

1. Ejecutar `CorRad` con las opciones:
   - `-w512`
   - `-h512`
   - `-ri_99(mala)512.raw`
   - `-ci_00(mala)512g.raw`[2]
   - `-ti_00(mala)512r.raw`
   - `-l`
   - `-p` (Si deseamos que `CorRad` guarde la imagen resultado en formato PPM)

```
D:\>CorRad -w512 -h512 -ri_99(mala)512.raw -ci_00(mala)512g.raw -ti_00(mala)512r.raw -l
CorRad: Radiometric correction 1.0
(C) Copyright 2000-2001. Vicente M. Arevalo Espejo

Radiometric correction
begin

        Equalize histogram ...... done

        Compute histogram ...... done

        Compute Inv(G) function ...... done

        Correct image ...... done

end
```

El fichero *log* que genera `CorRad` es el siguiente:

```
// Radiometric correction
// **********************
// begin
[Radiometric correction]
[Inv(G) function]
0
0
0
0
```

[2]Corregida geométricamente.

```
0
0
0
0
0
0
0
0
96
96
97
97
98
98
98
99
99
100
100
100
100

.......

215
216
217
219
220
222
223
226
227
229
231
232
235
238
239
243
245
248
251
251
251
251
255
255
255
//  end
```

La imagen corregida radiométricamente por `CorRad` es la siguiente:

Figura B.20: Imagen corregida radiométricamente.

Módulo DifIma

El módulo `DifIma` realiza la diferencia de una imagen (imagen a corregir) con respecto a otra (imagen de referencia). Permite la posibilidad de obtener la imagen diferencia en valor absoluto o desplazada un determinado nivel de gris (offset).

Modo de operación

Ejemplo 1: Ayuda

1. Ejecutar `DifIma` con la opción **-?**

```
D:\>DifIma -?
DifIma: Images difference 1.0
(C) Copyright 2000-2001. Vicente M. Arevalo Espejo

Usage: DifIma -?|-a|-oOffset -wWidth -hHeight -rRefImg -cCorImg -tOutImg [-l] [-p]
        -?:     Display this information
        -a:     Absolute
        -oOffset:       Offset
        -wWidth:        Image width (Width % 128 = 0)
        -hHeight:       Image height (Height % 128 = 0)
        -rRefImg:       Reference image
        -cCorImg:       Image to correct
        -tOutImg:       Result image
        -l:     Log file
        -p:     Save result image to PPM format
```

Ejemplo 2: Diferencia de imágenes

1. Ejecutar `DifIma` con las opciones:

   - `-a` (Valor absoluto)
   - `-w512`
   - `-h512`
   - `-ri_99(mala)512.raw`
   - `-ci_00(mala)512r.raw`[3]
   - `-tresult.raw`
   - `-l`
   - `-p` (Si deseamos que `DifIma` guarde la imagen resultado en formato PPM)

```
D:\>DifIma -a -w512 -h512 -ri_99(mala)512.raw -ci_00(mala)512r.raw -tresult.raw -l
DifIma: Images difference 1.0
(C) Copyright 2000-2001. Vicente M. Arevalo Espejo

Changes detection
begin

        Detect changes ..... done

end
```

El fichero *log* que genera `DifIma` es el siguiente:

```
// Changes detection
// *****************
// begin
[Changes detection]
// 0: DNd = abs(DNr - DNc)
// 1: DNd = (DNr - DNc + Offset)
[Mode]
0
//  end
```

La imagen diferencia procesada por `DifIma` es la siguiente:

[3] Corregida radiométricamente.

Figura B.21: Imagen diferencia procesada (-a).

Módulo DetCam

El módulo `DetCam` realiza el proceso de detección de cambios completo: corrección geométrica, radiométrica y diferencia de imágenes de una imagen (imagen a corregir) con respecto a otra (imagen de referencia). Al igual que ocurría con `DifIma`, permite la posibilidad de obtener la imagen diferencia en valor absoluto o desplazada un determinado nivel de gris (offset).

Modo de operación

Ejemplo 1: Ayuda

1. Ejecutar `DetCam` con la opción **`-?`**

```
D:\>DetCam -?
DetCam: Changes detection 1.0
(C) Copyright 2000-2001. Vicente M. Arevalo Espejo

Usage: DetCam -?|-a|-oOffset -wWidth -hHeight -rRefImg -cCorImg -tOutImg [-l] [-i] [-p] [-yPolygon]
        -?:     Display this information
        -a:     Absolute (Images difference)
        -oOffset:       Offset (Images difference)
        -wWidth:        Image width (Width % 128 = 0)
        -hHeight:       Image height (Height % 128 = 0)
        -rRefImg:       Reference image
        -cCorImg:       Image to correct
        -tOutImg:       Result image
        -l:     Log file
        -i:     Intermediate images
        -p:     Save result image to PPM format
        -yPolygon:      Polygon of interest (Geometric correction)
```

Ejemplo 2: Detección de cambios

1. Ejecutar `DetCam` con las opciones:
   - `-o127` (Offset de 127)
   - `-w512`
   - `-h512`
   - `-ri_99(mala)512.raw`
   - `-ci_00(mala)512.raw`
   - `-tresult.raw`
   - `-l`
   - `-i`
   - `-p` (Si deseamos que `DetCam` guarde la imagen resultado en formato PPM)
   - `-yMalaga.pol`

```
D:\>DetCam -o127 -w512 -h512 -ri_99(mala)512.raw -ci_00(mala)512.raw -tresult.raw -l -i -yMalaga.pol
DetCam: Changes detection 1.0
(C) Copyright 2000-2001. Vicente M. Arevalo Espejo

Geometric correction
begin

        Compute GCPs ..... done

        Remove bad GCPs ..... done

        Compute coefficients ..... done

        Correct image ..... done

end

Radiometric correction
begin

        Equalize histogram ..... done

        Compute histogram ..... done

        Compute Inv(G) function ..... done

        Correct image ..... done

end

Changes detection
```

```
begin

        Detect changes ..... done

end
```

El fichero *log* que genera `DetCam` es el siguiente:

```
// Geometric correction
// ********************
// begin
[Geometric correction]
[Tile width]
128
[Tile height]
128
[X tile number]
4
[Y tile number]
4
[Tile number]
16
// KLT algorithm
[Number of features]
25
[Window width]
15
[Window height]
15
// -1: Bad tile; -2: Tile out of polygon
// Tile.y, Tile.x, R.x, R.y, C.x, C.y, CX, CY
[Tracked points]
0   0   -2.000000   -2.000000   -2.000000   -2.000000   0.000000    0.000000
0   1   -2.000000   -2.000000   -2.000000   -2.000000   0.000000    0.000000
0   2   -2.000000   -2.000000   -2.000000   -2.000000   0.000000    0.000000
0   3   -2.000000   -2.000000   -2.000000   -2.000000   0.000000    0.000000
1   0   -1.000000   -1.000000   -1.000000   -1.000000   0.000000    0.000000
1   1   221.000000  159.000000  205.408981  170.568642  15.591019   -11.568642
1   2   357.000000  165.000000  341.301376  173.089973  15.698624   -8.089973
1   3   477.000000  157.000000  461.327713  165.482704  15.672287   -8.482704
2   0   53.000000   286.000000  38.073761   297.015652  14.926239   -11.015652
2   1   -1.000000   -1.000000   -1.000000   -1.000000   0.000000    0.000000
2   2   -1.000000   -1.000000   -1.000000   -1.000000   0.000000    0.000000
2   3   444.000000  286.000000  428.203423  294.332935  15.796577   -8.332935
3   0   -2.000000   -2.000000   -2.000000   -2.000000   0.000000    0.000000
3   1   -2.000000   -2.000000   -2.000000   -2.000000   0.000000    0.000000
3   2   -2.000000   -2.000000   -2.000000   -2.000000   0.000000    0.000000
3   3   -2.000000   -2.000000   -2.000000   -2.000000   0.000000    0.000000
[Number of tracked points]
5
// Tile.y, Tile.x, R.x, R.y, C.x, C.y, CX, CY
[Deleted points]
[End]
[Number of deleted points]
0
// x' = a0 + a1 * x + a2 * y + a3 * x * y
// y' = b0 + b1 * x + b2 * y + b3 * x * y
[Coefficients]
16.478103   0.997803    -0.005512   0.000015
-16.520279  0.019285    1.017649    -0.000041
//  end
```

```
// Radiometric correction
// **********************
// begin
[Radiometric correction]
[Inv(G) function]
0
0
0
0
0
0
0
0
0
0
0
96
96
97
97
98
98
98
99
99
99
100
100
101

........

215
216
217
219
220
221
224
226
227
229
231
233
235
238
240
243
245
248
251
251
251
251
255
255
255
//  end

// Changes detection
```

```
// *****************
// begin
[Changes detection]
// 0: DNd = abs(DNr - DNc)
// 1: DNd = (DNr - DNc + Offset)
[Mode]
1
//  end
```

La imagen diferencia procesada por `DetCam` es la siguiente:

Figura B.22: Imagen diferencia procesada (-o127).

Las imágenes intermedias son idénticas a las obtenidas en procesos anteriores, figuras B.19 (corrección geométrica) y B.20 (corrección radiométrica).

B.4.2 Interfaz gráfico

En este apartado realizaremos una descripción pormenorizada del funcionamiento del interfaz gráfico[4].

Modo de operación

Los pasos necesarios para llevar a cabo los distintos procesos son los siguientes:

[4]DetCam Interface 1.0 es un *front end* para los módulos descritos anteriormente.

Escritorio

Situarse en el *Escritorio de Windows 98* (ver figura B.23). Las operaciones a realizar son las siguientes:

1. Hacer *click* en el botón `Inicio`
2. Hacer *click* en el icono `Programas`
3. Hacer *click* en el icono `DetCam Interface 1.0`

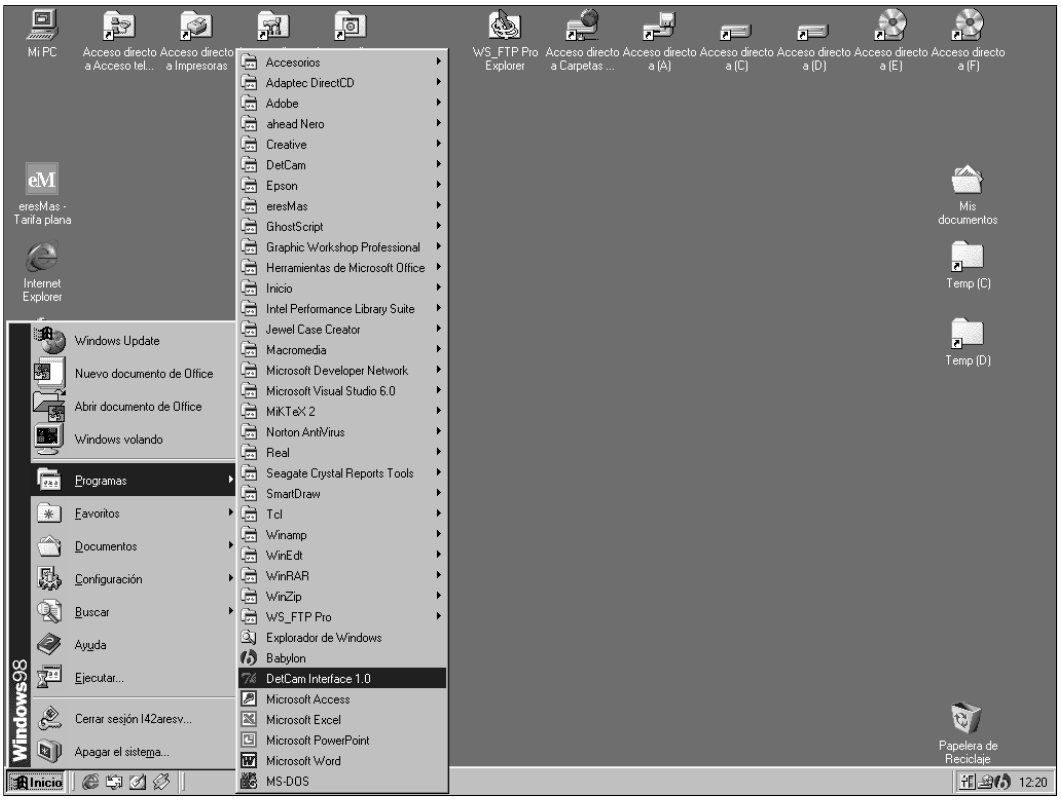


Figura B.23: Escritorio de Windows 98.

Tras realizar estas operaciones se visualiza la ventana *DetCam Interface 1.0*.

DetCam Interface 1.0

Situarse en la ventana *DetCam Interface 1.0* (ver figura B.24). Esta ventana constituye el punto de entrada del interfaz, contiene los datos comunes a todos los programas: imagen de referencia, imagen a corregir, ancho y alto, etc. Las operaciones a realizar son las siguientes:

1. Completar las cajas de texto con los datos adecuados:

   (a) Completar la caja `Reference image (RAW)` con la imagen de referencia, bien escribiendo directamente o pulsando en el botón anexo <<.

   (b) Completar la caja `Image to correct (RAW)` con la imagen a corregir, bien escribiendo directamente o pulsando en el botón anexo <<.

   (c) Completar la caja `Result image (RAW)` con el nombre de la imagen de salida. `DetCam Interface 1.0` permite la posibilidad de indicar el el directorio destino pulsando en el botón anexo <<.

   (d) Completar las cajas `Width` y `Height` con el ancho y alto de las imágenes de entrada.

   (e) Hacer *click* en la opción `Generate log file` si desea que los programas generen el fichero *log*.

   (f) Hacer *click* en la opción `Save result image to PPM` si desea que los programas guarden la imagen de salida en formato PPM.

2. Elegir la opción adecuada del submenú `Program`

   NOTA: Cuando pulsamos el botón << realizamos una llamada a la API de Windows que abre la ventana *Abrir* (ver figura B.25).

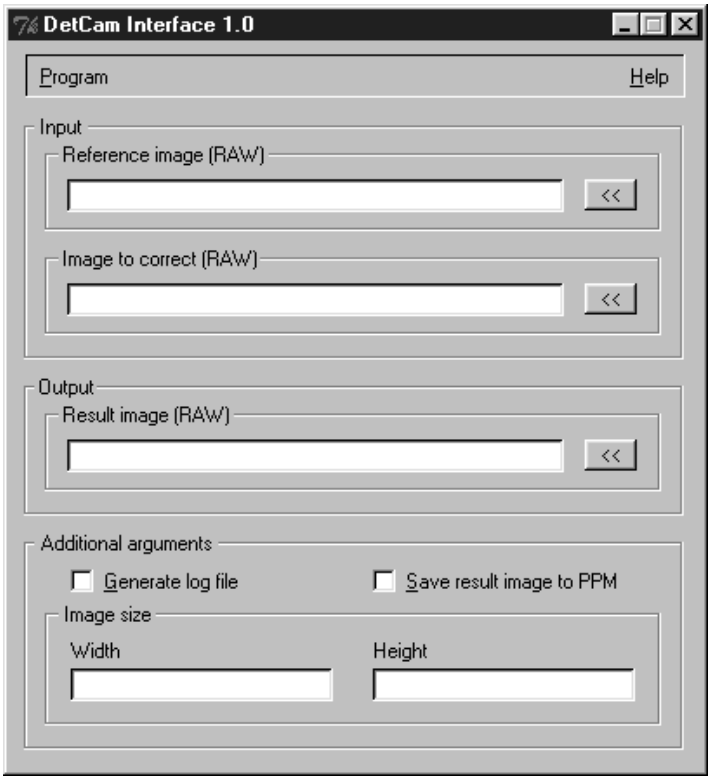


Figura B.24: DetCam Interface 1.0.

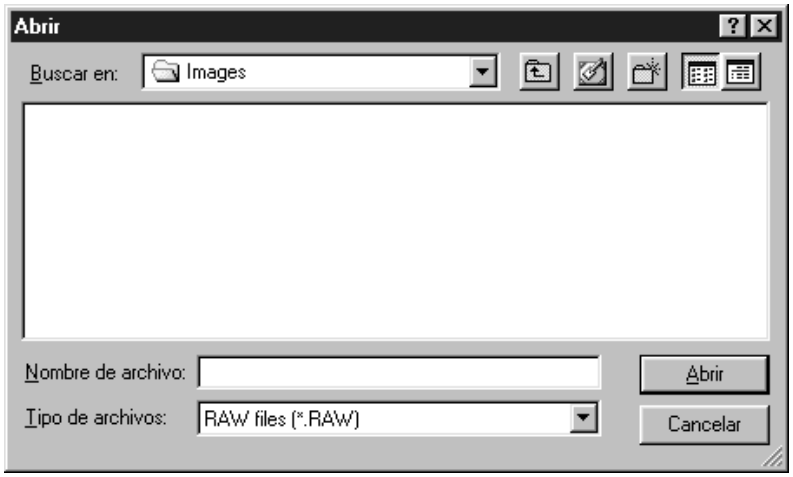


Figura B.25: Abrir RAW file.

Opción 1

Se selecciona la opción del menú `Geometric correction`. Las operaciones a realizar son las siguientes:

1. Hacer *click* en la opción `Program` del menú principal
2. Hacer *click* en la opción `Geometric correction` del submenú `Program`

Tras realizar estas operaciones se visualiza la ventana *Geometric correction*.

Opción 2

Se selecciona la opción del menú `Radiometric correction`. Las operaciones a realizar son las siguientes:

1. Hacer *click* en la opción `Program` del menú principal
2. Hacer *click* en la opción `Radiometric correction` del submenú `Program`

Tras realizar estas operaciones se visualiza la ventana *Radiometric correction*.

Opción 3

Se selecciona la opción del menú `Images difference`. Las operaciones a realizar son las siguientes:

1. Hacer *click* en la opción `Program` del menú principal
2. Hacer *click* en la opción `Images difference` del submenú `Program`

Tras realizar estas operaciones se visualiza la ventana *Images difference*.

Opción 4

Se selecciona la opción del menú `Changes detection`. Las operaciones a realizar son las siguientes:

1. Hacer *click* en la opción `Program` del menú principal
2. Hacer *click* en la opción `Changes detection` del submenú `Program`

Tras realizar estas operaciones se visualiza la ventana *Changes detection*.

Geometric correction

Situarse en la ventana *Geometric correction* (ver figura B.26). Esta ventana es la encargada de lanzar `CorGeo` en *background* y presentar los resultados, contiene los datos específicos de este programa: polígono de interés, etc. Las operaciones a realizar son las siguientes:

1. Hacer *click* en la opción `Polygon file` si desea especificar una región de interés. Completar la caja con el fichero que contiene el POI, bien escribiendo directamente o pulsando en el botón anexo <<

2. Hacer *click* en el botón `Run` para ejecutar `CorGeo`

3. Hacer *click* en el botón `View log file` si desea ver el fichero *log* generado

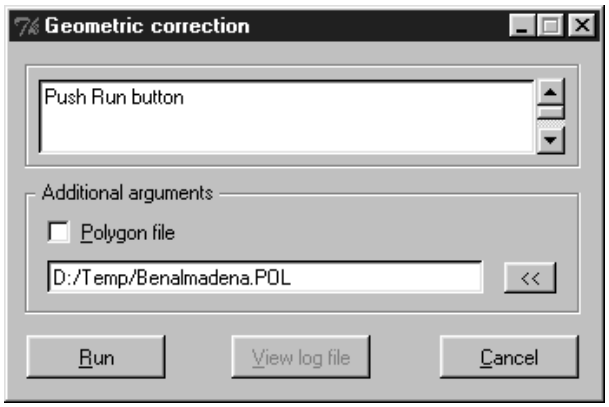


Figura B.26: Geometric correction.

NOTA: Si la opción `Save result image to PPM` ha sido activada la imagen resultado es visualizada automáticamente, una vez finalizado el proceso de corrección.

Cuando pulsamos el botón << realizamos una llamada a la API de Windows que abre la ventana *Abrir* (ver figura B.27).

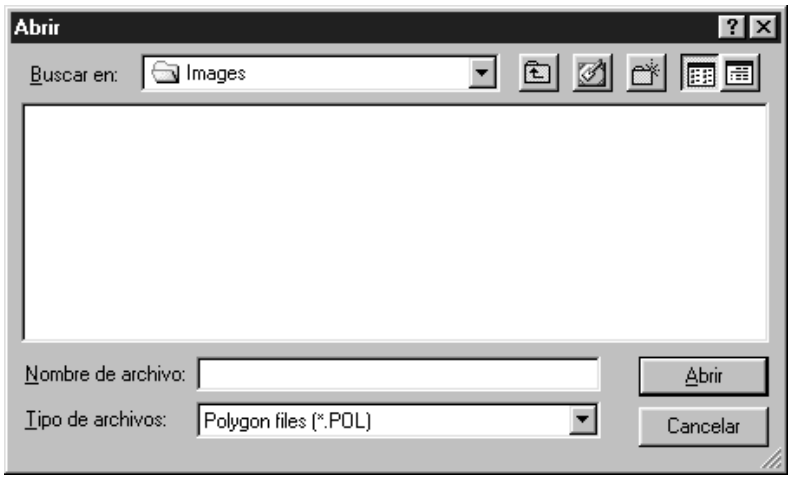


Figura B.27: Abrir POL file.

Ejemplo 1: Corrección geométrica

1. Ejecutar `CorGeo` en *background* con las opciones:
   - `-w512`
   - `-h512`
   - `-ri_00(benal)512.raw`
   - `-ci_99(benal)512.raw`
   - `-ti_99(benal)512g.raw`
   - `-l`
   - `-p`
   - `-yBenalmadena.pol`

Para ello es necesario completar la ventana `DetCam Interface 1.0` con los datos adecuados, estos son:

1. `Width`: 512
2. `Height`: 512
3. `Reference image (RAW)`: i_00(benal)512.raw
4. `Image to correct (RAW)`: i_99(benal)512.raw

5. `Result image (RAW)`: i_99(benal)512g.raw

6. `Generate log file`: TRUE

7. `Save result image to PPM`: TRUE

A continuación, completamos el resto de parámetros en la ventana `Geometric correction`, estos son:

1. `Polygon file`: TRUE

2. `Polygon file (textbox)`: Benalmadena.pol

El fichero *log* que visualiza `DetCam Interface 1.0` es el siguiente:

Figura B.28: Fichero *log*.

La imagen corregida geométricamente que visualiza `DetCam Interface 1.0` es la siguiente:

Figura B.29: Imagen corregida geométricamente.

Radiometric correction

Situarse en la ventana *Radiometric correction* (ver figura B.30). Esta ventana es la encargada de lanzar `CorRad` en *background* y presentar los resultados. Las operaciones a realizar son las siguientes:

1. Hacer *click* en el botón `Run` para ejecutar `CorRad`
2. Hacer *click* en el botón `View log file` si desea ver el fichero *log* generado

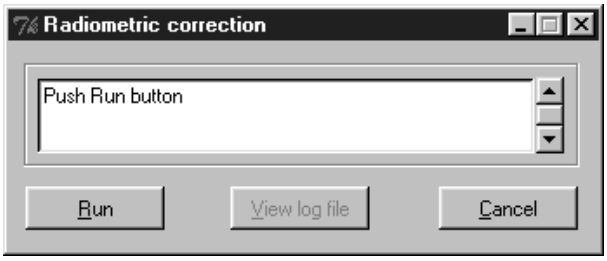


Figura B.30: Radiometric correction.

Ejemplo 1: Corrección radiométrica

1. Ejecutar `CorRad` en *background* con las opciones:

- `-w512`
- `-h512`
- `-ri_00(benal)512.raw`
- `-ci_99(benal)512g.raw`
- `-ti_99(benal)512r.raw`
- `-l`
- `-p`

Para ello es necesario completar la ventana `DetCam Interface 1.0` con los datos adecuados, estos son:

1. `Width`: 512
2. `Height`: 512
3. `Reference image (RAW)`: i_00(benal)512.raw
4. `Image to correct (RAW)`: i_99(benal)512g.raw
5. `Result image (RAW)`: i_99(benal)512r.raw
6. `Generate log file`: TRUE
7. `Save result image to PPM`: TRUE

El fichero *log* que visualiza `DetCam Interface 1.0` es el siguiente:

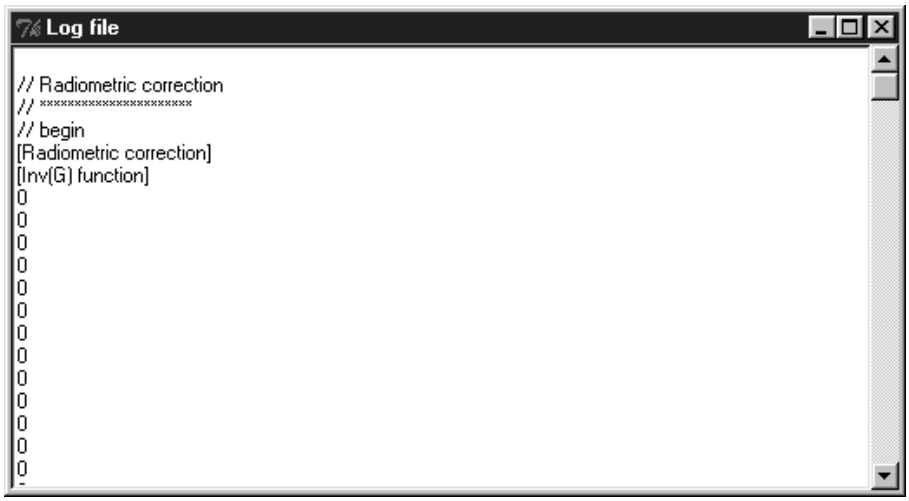


Figura B.31: Fichero *log*.

La imagen corregida radiométricamente que visualiza `DetCam Interface 1.0` es la siguiente:

Figura B.32: Imagen corregida radiométricamente.

Images difference

Situarse en la ventana *Images difference* (ver figura B.33). Esta ventana es la encargada de lanzar `DifIma` en *background* y presentar los resultados,

contiene los datos específicos de este programa: tipo de procesamiento (valor absoluto, desplazamiento), etc. Las operaciones a realizar son las siguientes:

1. Hacer *click* en la opción `DNout = abs(DNref - DNcor)` si desea que se le aplique el valor absoluto a la imagen diferencia
2. Hacer *click* en la opción `DNout = DNref - DNcor + Offset` si desea que se desplace la imagen diferencia un determinado nivel de gris
3. Hacer *click* en el botón `Run` para ejecutar `DifIma`
4. Hacer *click* en el botón `View log file` si desea ver el fichero *log* generado

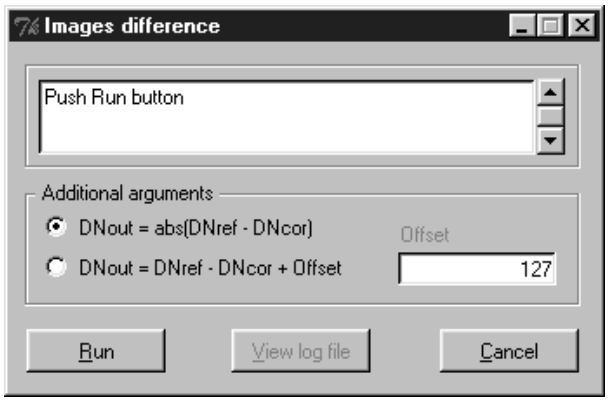


Figura B.33: Images difference.

Ejemplo 1: Diferencia de imágenes

1. Ejecutar `DifIma` en *background* con las opciones:
   - `-o127`
   - `-w512`
   - `-h512`
   - `-ri_00(benal)512.raw`
   - `-ci_99(benal)512r.raw`
   - `-tResult.raw`

- `-l`
- `-p`

Para ello es necesario completar la ventana `DetCam Interface 1.0` con los datos adecuados, estos son:

1. `Width`: 512
2. `Height`: 512
3. `Reference image (RAW)`: i_00(benal)512.raw
4. `Image to correct (RAW)`: i_99(benal)512r.raw
5. `Result image (RAW)`: Result.raw
6. `Generate log file`: TRUE
7. `Save result image to PPM`: TRUE

A continuación, completamos el resto de parámetros en la ventana `Images difference`, estos son:

1. `DNout = DNref - DNcor + Offset`: TRUE
2. `DNout = DNref - DNcor + Offset (textbox)`: 127

El fichero *log* que visualiza `DetCam Interface 1.0` es el siguiente:

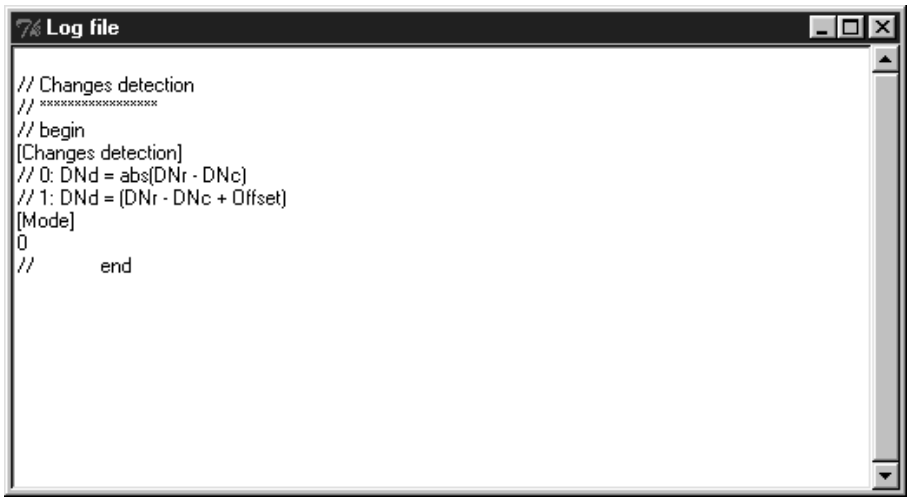


Figura B.34: Fichero *log*.

La imagen diferencia procesada que visualiza `DetCam Interface 1.0` es la siguiente:

Figura B.35: Imagen diferencia procesada.

Changes detection

Situarse en la ventana *Changes detection* (ver figura B.36). Esta ventana es la encargada de lanzar `DetCam` en *background* y presentar los resultados, contiene los datos específicos de este programa: tipo de procesamiento, polígono

de interés, etc. Las operaciones a realizar son las siguientes:

1. Hacer *click* en la opción `DNout = abs(DNref - DNcor)` si desea que se le aplique el valor absoluto a la imagen diferencia

2. Hacer *click* en la opción `DNout = DNref - DNcor + Offset` si desea que se desplace la imagen diferencia un determinado nivel de gris

3. Hacer *click* en la opción `Polygon file` si desea especificar una región de interés. Completar la caja con el fichero que contiene el POI, bien escribiendo directamente o pulsando en el botón anexo <<

4. Hacer *click* en la opción `Remove intermediate images` si desea eliminar las imágenes intermedias (las procesadas geométrica y radiométricamente)

5. Hacer *click* en el botón `Run` para ejecutar `CorGeo`

6. Hacer *click* en el botón `View log file` si desea ver el fichero *log* generado.

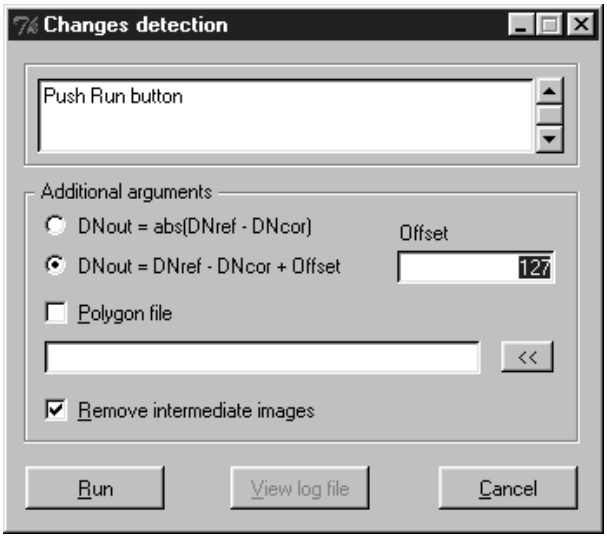


Figura B.36: Changes detection.

Ejemplo 1: Detección de cambios

1. Ejecutar `DetCam` en *background* con las opciones:
   - `-o127`
   - `-w512`
   - `-h512`
   - `-ri_00(benal)512.raw`
   - `-ci_99(benal)512.raw`
   - `-tResult.raw`
   - `-l`
   - `-p`

Para ello es necesario completar la ventana `DetCam Interface 1.0` con los datos adecuados, estos son:

1. `Width`: 512
2. `Height`: 512
3. `Reference image (RAW)`: i_00(benal)512.raw
4. `Image to correct (RAW)`: i_99(benal)512.raw
5. `Result image (RAW)`: Result.raw
6. `Generate log file`: TRUE
7. `Save result image to PPM`: TRUE

A continuación, completamos el resto de parámetros en la ventana `Changes detection`, estos son:

1. `DNout = DNref - DNcor + Offset`: TRUE
2. `DNout = DNref - DNcor + Offset (textbox)`: 127
3. `Remove intermediate images`: TRUE

El fichero *log* que visualiza `DetCam Interface 1.0` es el siguiente:

Figura B.37: Fichero *log*.

La imagen diferencia procesada que visualiza `DetCam Interface 1.0` es la siguiente:

Figura B.38: Imagen diferencia procesada.

B.5 Desinstalación

Para desinstalar la aplicación (programas, interfaz gráfico y ejemplos) en un *PC compatible* bajo *Windows 98* es necesario realizar los siguientes pasos:

1. Ejecutar `Uninstall.exe` desde el *Panel de control* (ejecución indirecta)

B.5.1 Modo de operación

Los pasos necesarios para llevar a cabo el proceso de desinstalación son los siguientes:

> **NOTA:** El programa puede ser desinstalado directamente a través del *Explorador de Windows*, accediendo al directorio especificado en el proceso de instalación y eliminando su contenido (ejecutables, directorio de ejemplos, etc.).
>
> Para completar el proceso de desinstalación es conveniente eliminar los enlaces directos del menú `Programas` de Windows.

Escritorio

Situarse en el *Escritorio de Windows 98* (ver figura B.39). Las operaciones a realizar son las siguientes:

1. Hacer *click* en el botón `Inicio`
2. Hacer *click* en el icono `Configuración`
3. Hacer *click* en el icono `Panel de control`

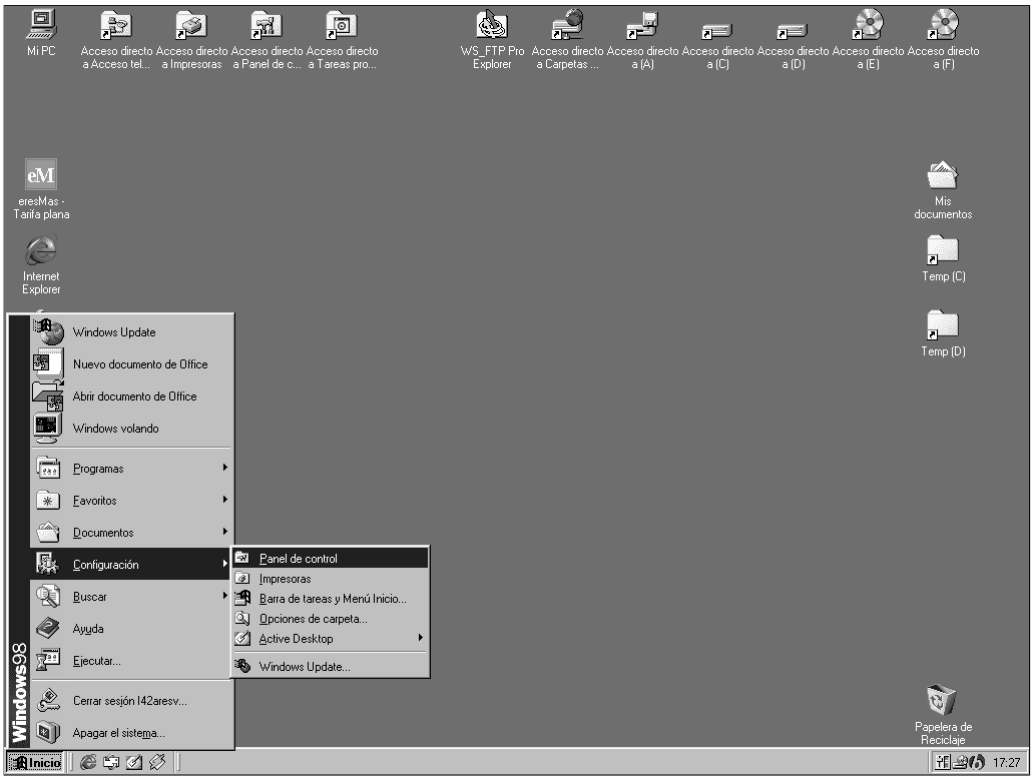


Figura B.39: Escritorio de Windows 98.

Tras realizar estas operaciones se visualiza la ventana *Panel de control*.

Panel de control

Situarse en la ventana *Panel de control* (ver figura B.40). Las operaciones a realizar son las siguientes:

1. Hacer *doble click* en el icono `Agregar o quitar programas`

Figura B.40: Panel de control.

Tras realizar estas operaciones se visualiza la ventana *Propiedades de Agregar o quitar programas*.

Propiedades de Agregar o quitar programas

Situarse en la ventana *Propiedades de Agregar o quitar programas* (ver figura B.41). Las operaciones a realizar son las siguientes:

1. Seleccionar la aplicación DetCam
2. Hacer *click* en el botón `Agregar o quitar`

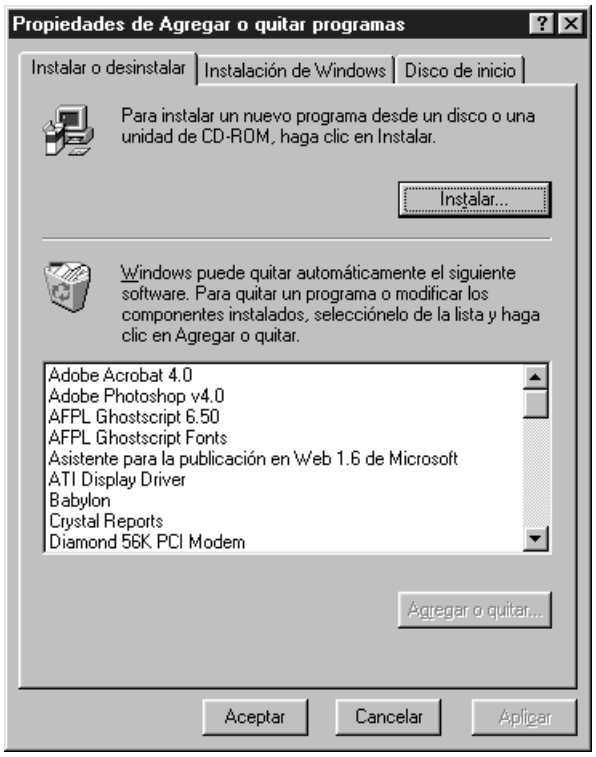


Figura B.41: Propiedades de Agregar o quitar programas.

Tras realizar estas operaciones se visualiza la ventana *Confirm file deletion*.

Confirm file deletion

Situarse en la ventana *Confirm file deletion* (ver figura B.42). Las operaciones a realizar son las siguientes:

1. Pulsar return (el botón por defecto es `Sí`) para continuar con el proceso de desinstalación

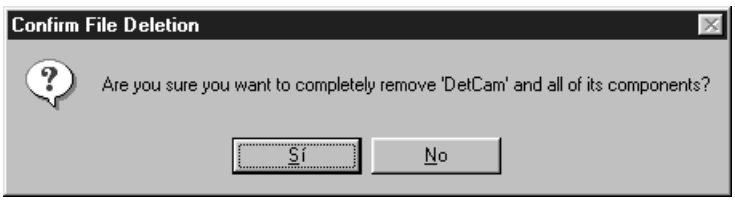


Figura B.42: Confirm file deletion.

Tras realizar estas operaciones se visualiza la ventana *Remove programs from your computer*.

Remove programs from your computer

Situarse en la ventana *Remove programs from your computer* (ver figura B.43). Las operaciones a realizar son las siguientes:

1. Pulsar return (el botón por defecto es OK) para finalizar el proceso de desinstalación

Figura B.43: Remove programs from your computer.

Tras realizar estas operaciones el proceso de desinstalación finaliza.