Classificador de textura para imagenes satellite

Contents

1. Introducción

Primero querrá decir que este documento no sera solamente un informe sino también un manual para utilizar mi programa y que cualquier alumno pueda seguir mi trabajo. Por eso hablare de mi manera de programar, de la estructura de mi programa para que uno pueda entenderlo y completarlo.
El objetivo final de este proyecto es de automatizar el tratamiento de imágenes satélites para predecir el riesgo de inundaciones en una zona de Santiago de Chile. Un objetivo ambicioso y después mi contribución faltara mucho mas trabajo para finalizar este proyecto.
La primera parte de este informe se concentrara en la estructura del programa, como es construido, los distintos archivos y sus conexiones entre ellos. Luego hablare de todos los problemas que he debido solucionar. En la segunda parte presentare la librería para leer los archivos HDF, es decir las imágenes satélites. La tercera hablara de mi manera para mostrar las imagénes en la pantalla y de herramientas básicas de tratamiento (Zoom, contraste, colores...). En cuarta parte hablare de lo que mas nos interesa : los clasificadores de texturas de una manera general, y después la quinta y ultima parte tratara de explicar los algoritmos de clasificador de textura.

2. Estructura del programa

La primera cosa que deben saber es que el GUI (Graphics User Interface) es hecho con la **librería Qt**.
En este parte les hablare de todos los archivos importantes que encontraran en mi programa.

2.1. MainWindow

Obviamente, como todo programa, este empieza con el archivo main.cpp abriendo la ventana principal : un QMainWindow(Widget de Qt). Más abajo una imagen de la ventana principal. Es compuesto de 3 partes :

1. Zona central
   Este parte permite de mostrar las imágenes satélites con pestañas permitiendo de abrir distintas imágenes al mismo tiempo.
2. Explorador de HDF Este parte es donde los distintos archivos contenidos en un HDF son presentados.
3. Herramientas Este parte que esta compuesta de 3 pestañas permite de hacer todos los tratamientos de imágen : modificar el contraste y la luminosidad, crear una imagen en color y clasificar las texturas.

De un punto de vista estructural hay que ver este archivo (MainWindow.cpp) como el centro del programa, todo los otros archivos pasan por este.

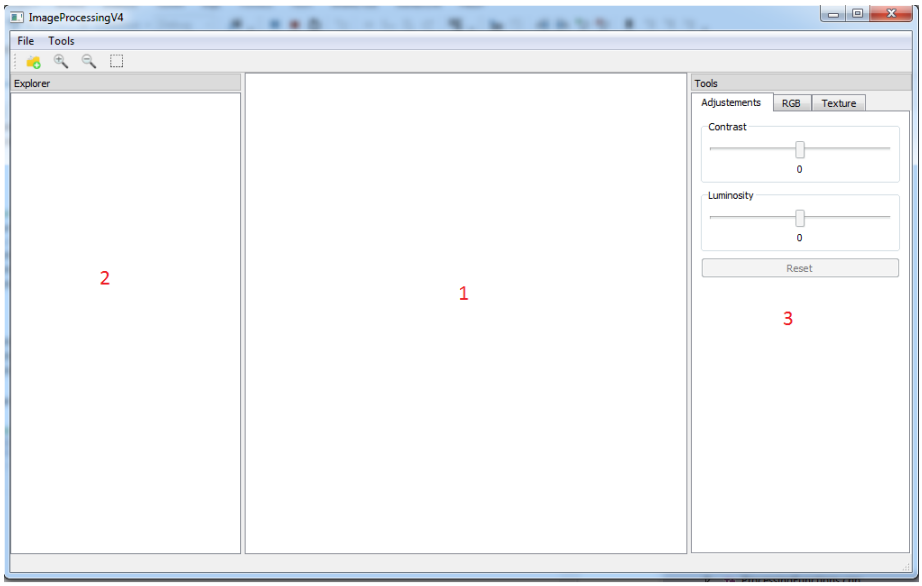

Figura 1: La ventana principal : (1)Zona Principal, (2)Explorador de HDF,(3)Herramientas

2.2. ExplorerWindow

Este archivo permite solamente de abrir los archivos HDF y mostrar su estructura en la parte "Explorer" de la ventana principal. Digo "solamente" pero verán que no fue tan facíl para llegar a este resultado. Es por eso que escribiré una parte entera sobre este problema.

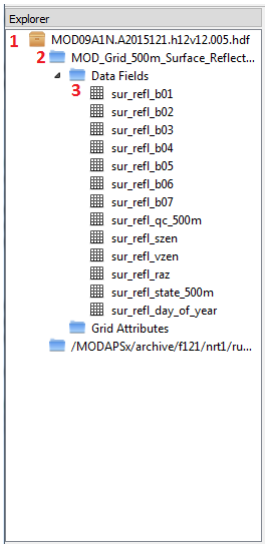

Figura 2: Explorador: (1)Archivo HDF, (2)Carpeta, (3)Imagen

2.3. OpenFile

No hablare mucho de este archivo en el informe. Permite solamente de crear un objeto cuando abrimos una imagen. En este objeto guardar muchas informaciones sobre la imagen : su nombre, de que archivo HDF viene, sus dimensiones, la estructura cv::Mat de la imagen, el coeficiente de contraste y luminosidad... Eso permite un tratamiento de los archivos abiertos mas facíl. También permite de guardar la estructura cv::Mat original antes de modificarla, y asi poder hacer un "reset".

2.4. MyGraphicView

Este archivo es uno de los mas importantes del programa : permite de mostrar la imagen. En efecto en la librería Qt la mejor forma de mostrar imágenes grandes es el QWidget QGraphicView. Pero para agregar herramientas como el Zoom he debido redefinir este Widget. Ademas para seleccionar una zona en la imagen (para el classificador de imagen) uso funciones definidas en este archivo. En resumen este archivo me permite de modificar como lo quierro el widget QGraphicView y así permitir al usuario de interactuar con la imagen.

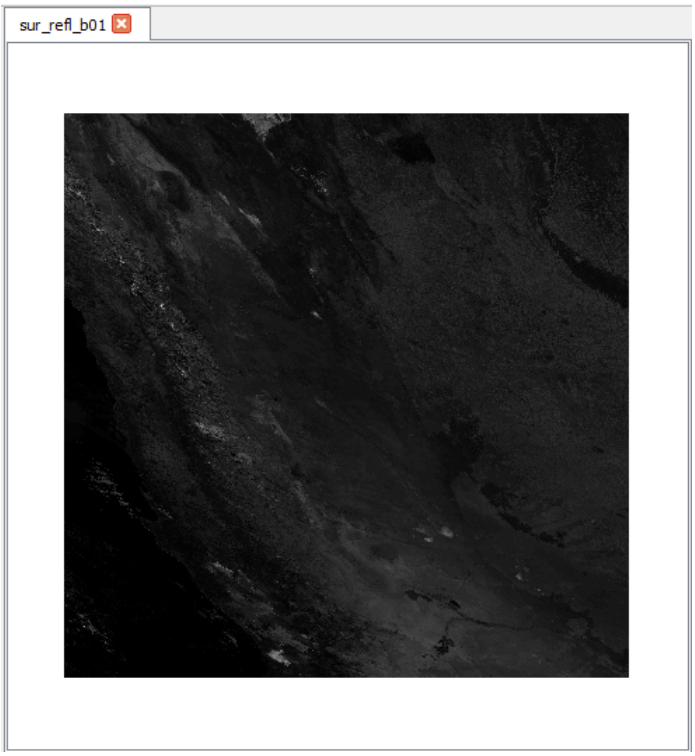

Figura 3: MyGraphicView

2.5. ToolsDock

Otra vez ToolsDock es un archivo esencial : permite todos los tratamientos de imágenes. Es compuesto de 3 pestañas llamadas : "Adjusments","RGB" y "Texture". Voy a describir los 3 con detalles.

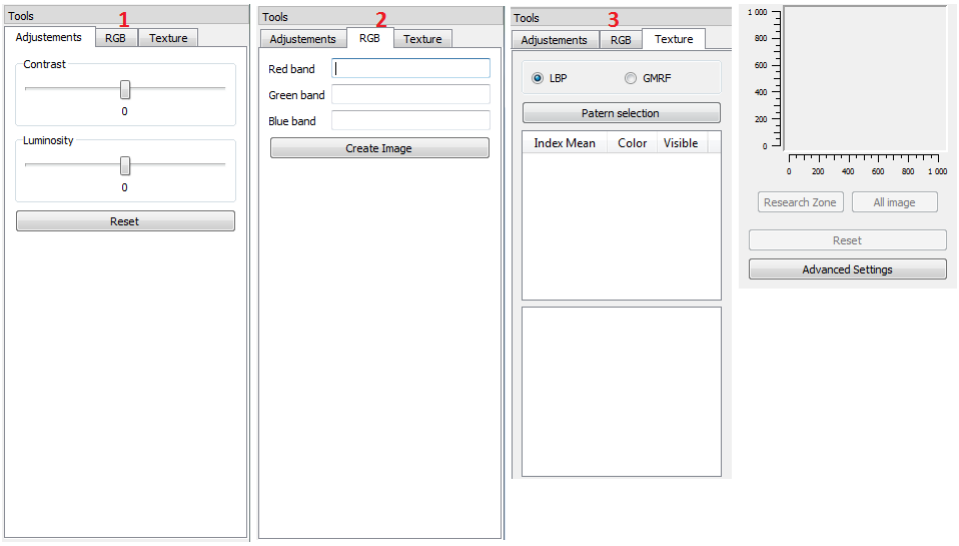

Figura 4: ToolsDock : (1)Cambiar contraste y luminosidad, (2)Crear una imagen en color,(3)Classificador de texturas

1. Adjustments Esta pestaña permite de cambiar el contraste y la luminosidad de una imagen abierta. Un otro archivo vinculado con este pestaña llamado MySlider me permitio de redefinir la clase "QSlider" para que las modificaciones sobre la imagen sean hechas solamente cuando el usuario suelta el botón del ratón.

2. RGB En esta pestaña pueden ver tres cuadros de texto que representan cada uno un componente de una imagen en color RGB (Red,Green,Blue). Para llenar estos cuadros hay que seleccionar una imagen en la ventana "Explorer" y deslizarlo hasta el cuadro. Luego pueden crear una imagen RGB presionando el botón "Create Image".

3. Texture Llegamos finalmente a la pestaña mas importante y tambien la más complicada. Primero hay que eligir el algoritmo de clasificador de textura : LBP (Local Binary Pattern) o GMRF (Gaussian Markov Random Field). Después eso pueden definir una zona de la imagen como referencia (training). Verán que una linea se agrega al la tabla con distintas columnas y un gráfico dependiendo del algoritmo eligido:

   - LBP : Columnas : Index Mean es el promedio del LBP (un numero binario) de cada pixel de la zona elegida ; Color es el color que tomaran los pixeles asignados a este "label" (o referencia), Visible permite de esconder el gráfico de este label. En el gráfico más abajo esta representado el histograma del LBP, veremos más adelante en este informe que es exactamente. Finalmente para iniciar el algoritmo de clasificación hay presionar el botón "Research Zone" y luego seleccionar la zona donde quieren clasificar las texturas.
   - GMRF : Columnas : Mean es el promedio de la intensidad de los pixeles en la zona elegida, Variance es la varianza esta misma intensidad. Con estos valores construimos la curva de Gauss en el gráfico mas abajo. Y Finalmente como el LBP hay que seleccionar la zona de búsqueda.

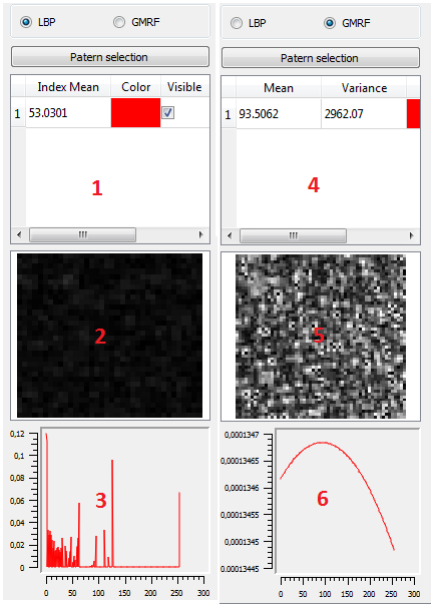

Figura 5: Textures : (1 y 4) Tablas donde se mostra las zonas de referencia (label), (2 y 5) Imagen de la zona de referencia seleccionada, (3) Histograma del LBP en la zona de referencia, (6) Curva de Gauss de las intensidades en la zona de referencia.

2.6. ProcessingFunctions

En este archivo esta definido funciones estáticas para el tratamiento de imágenes : contraste y luminosidad y clasificador de texturas. En resumen la ventana "ToolsDock" hace todos los cálculos con funciones que están en este archivo.

En resumen abajo un esquema que mostra los vínculos entre los distintos archivos.

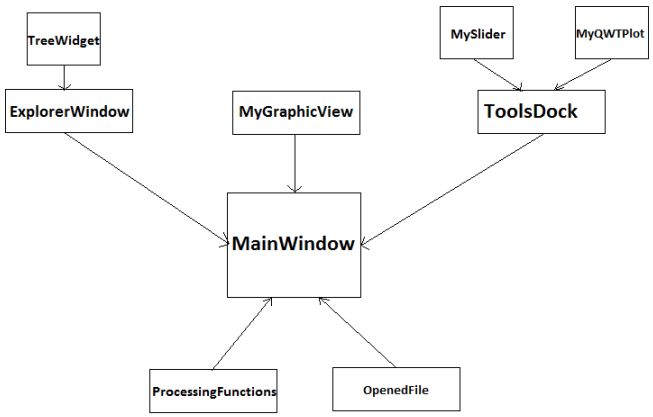

Figura 6: Estructura del programa

3. Los archivos HDF

Llegamos a la parte de los archivos HDF, aunque abrir una imagen puede parecer muy facíl no es el caso de los HDF. HDF significa Hierarchical Data Format, es un archivo creado para contener distintos archivos de una manera jerarquizada (con carpetas).

3.1. Problemas generales sobre la librería

Existe dos versiones de HDF : HDF4 y HDF5. Las imágenes satelites que tenemos son guardadas en archivos HDF4. En la documentación de la librería dedicada a manejar los HDF dicen que es preferible de usar la versión 5. Entonces mi primera intento fue de instalar la librería H4TOH5 para convertir nuestros archivos y despues usar solamente la librería HDF5. Pero nunca consigue a instalar esta librerías (H4TOH5) y muchas personas tienen el mismo problema en los foros. Puede ser que la librería H4TOH5 no se puede compilar con Visual Studio. Finalmente decidí de seguir con los archivos en HDF4. En cambio, esta librería no parece actualizada frecuentemente y entonces tuve problemas para manejarla.

3.2. Explicaciónes para abrir un HDF

Para mí esta etapa fue bastante dificíl porque hay muy poca documentación on internet y el proceso completo no se puede conseguir todo hecho. Por eso querría entrar en los detalles de como abrir un HDF.

1. Abrir el HDF :
   La primera etapa es de abrir el archivo con la función :

   Hopen*([Nombre del archivo], [Modo de acceso], [num_dds_block]));*

   El Modo de acceso especifica si querremos leer, escribir o ambos el archivo. Aún no entendi el ultimó argumento. Ver la documentación para saber más.

2. Obtener el nombre de todas las carpetas a la raíz del HDF :
   Para eso usamos la función :

   Vlone*([File_id],[Tabla],[Numero de carpetas]);*

   El File_id es el valor retornado por la función Hopen.La tabla es una tabla que va a contener los IDs de todas la carpetas, de tamaño igual al numero de carpetas. Y para tener el numero de carpeta tenemos que llamar la misma función pero con los argumentos siguientes:

   Vlone*([File_id,NULL,[Variable donde se guardara el numero de carpeta]);*

3. Buscar en cada carpeta si hay otras carpetas o otro tipo de archivo :
   Primero tenemos que vincularse a una de las carpetas a la raíz del HDF con la función :

   Vattach*([file_id],[Tabla de las carpetas][i],"r");*

   Creo que esta función es facíl de entender, lo único que tengo que decir que es que el valor retornado por esta función es el ID de este carpeta. Y necesitamos de este ID para la próxima etapa : obtener el nombre de de las carpetas a la raíz con la función :

Vgetname*([ID retornado por la función Vattach],[Nombre])*;

También un función muy facíl de entender : con el ID de la carpeta, guardamos su nombre en la variable [Nombre].
Hasta acá lo que tenemos es :

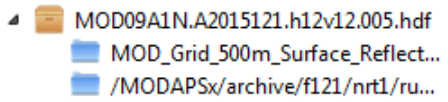

Figura 7: Carpetas a la raíz del HDF

4. Buscar otras carpetas:
   En cada carpeta a la raíz del HDF hay otras carpetas llamados en la librería HDF4 Vgroups. Entonces en cada carpeta a la raíz buscamos los VGroups :

   - Tenemos el numero de Vgroups con la función :

     Vgetvgroups*([Vgroup_id],[start_Vgroup],[count_Vgroup,NULL);*

     Donde Vgroup_id es el valor retornado por la función Vattach, start_Vgroup el "index" del primero Vgroup que querremos buscar(aquí los querremos todos entonces startVgroup $= 0$), count_Vgroup el numero de Vgroup que querremos buscar (aqui 0 porque usamos aquí esta función para que nos da este numero).
   - Buscar los Id de los Vgroups con la misma función :

     Vgetvgroups*([Vgroup_id],0,[count_Vgroup],[tabla]);*

     Donde count_Vgroup es igual al valor retornado por la función anterior, y tabla es donde la función guarda los ID.

   Después eso podemos tener el nombre de los Vgroups de la misma manera que hemos obtenido el nombre de las carpetas a la raíz. Y Obviamente en cada Vgroup puede haber otros Vgroups, entonces tenemos que hacer una vuelta hasta que no hay más Vgroup en todo el archivo. Una vez eso hecho tenemos lo siguiente :

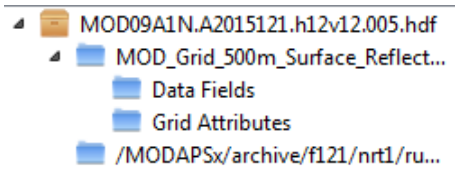

Figura 8: Todas las carpetas en el HDF

5. Buscar archivos: La próxima etapa es de encontrar los archivos guardados en el HDF. Existen distintos tipos de archivos pero los dos importantes (los que nos interesan ahora) son :

   - VDatas :
     Una VData es un tipo de archivo que permite de guardar cualquier tipo de tablas. Los tipos de variables pueden ser cualquier cosa y por eso leer este tipo de archivo es complicado. Hice un algoritmo que permite de leer algunos archivos VData pero no todos, por eso no lo deje en el

programa pero pueden encontrar el algoritmo par buscarlas y para leerlas en los anexos. Si lo quieren implementar necesitaran agregar algunas cosas para poder leer todos los tipos (cambiar los tipos de variables).

- SDS :
  SDS significa "Scientific Data Set" y es este tipo de archivo que nos interesa especialmente. En efecto contiene las imágenes satélites. Como las Vdatas los SDS pueden contener distintos tipos de variables :
  - 16 bits integer
  - unsigned 8 bits integer
  - unsigned 16 bits integer

  Esto cambio de variable puede ser difícil de manejar, en efecto par guardar en una tabla este archivo, hay que cambiar el tipo de variable. En seguida vamos a ver como el programa lee los SDS.

La primera etapa para buscar los archivos es de conocer el numero de archivos que hay en el VGroup donde "estamos" :

int32 n_obj = **Vntagrefs***([vgroup_id]);*

Luego podemos buscar todos los archivos que hay en un Vgroup con:

Vgettagref*([vgroup_id], [obj_index], [&obj_tag], [&obj_ref]);*

En un vgroup cada "objeto" tiene un indice que va de 0 a n_obj. El problema en eso es que un objeto puede ser cualquier cosa : vgroup, vdata a SDS. Entonces hay que separar los tres. Aqui buscamos los SDS entonces primero averiguamos que no es un vgroup con la función :

Visvg*([group_id], [obj_ref]*

Si es un vgroup la función retorna un "true". Y para saber si es un SDS y no una vdata usamos la función :

VSattach*([File Name],[obj_ref],"r");*

Si el archivo designado por [obj_ref] es un SDS la función retorna -1. Luego necesitamos el nombre del archivo. Aquí se complica un poco, según yo la libreria HDF4 es muy mal hecha para manejar este problema.

sd_id = **SDstart***([File Name], [Modo de acceso]);*
int32 sd_index = **SDreftoindex***([sd_id],[obj_ref]);*
int32 sds_id = **SDselect***([sd_id], [sd_index]);*
SDgetinfo*([sds_id], [name], [&rank], [&dim_sizes], [&data_type],[&n_attrs]);*

Arriba todas las commandas que tenemos que llamar solamente para tener el nombre y algunas información sobre el archivo. No voy a explicar todo, creo que es bastante facíl de entender. Solamente la función SDgetinfo nos da : el nombre, la dimensión de la tabla (en el caso de un imagen : 2)y el tipo de variable. Una vez que tenemos todo esto listo podemos empezar a leer los archivos. Al final obtenemos lo siguiente :

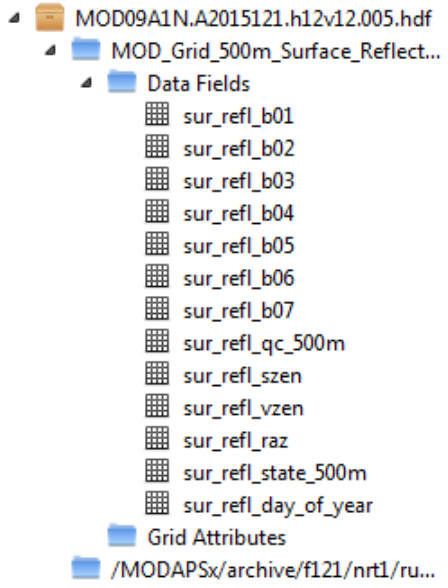

Figura 9: Apertura de un HDF completa

6. Leer los SDS : La primera etapa para leer este tipo de archivo es de iniciar el interfaz SD para el HDF abierto.

   int32 sd_id = **SDstart** *([FILE NAME],[MODO DE ACCESO]);*

   Creo que esta función es muy facíl de entender. Con ella sacamos el id de la interfaz SD que nos va a servir para la seguida. En efecto una vez que la interfaz esta "abierta" necesitamos seleccionar cual archivo SDS querremos leer. Después necesitamos vincularse a un SDS en particular diciendo su "sd_index" que tenemos de la función SDreftoindex().

   int32 sds_id = **SDselect***([sd_id],[sd_index]);*

   Ahora necesitamos guardar la tabla que esta en el archivo SDS en un tabla de nuestro programa. Hacemos eso con la funciíon :

   SDreaddata*([sds_id], [start], NULL, [edges], [data]);*

   De nuevo según yo esta librería no es actualizada y entonces esta funcíon no funciona como debería funcionar según el manual. En efecto, según este podemos dar una tabla de 2 dimensiones (o más) a la funcíon SDreaddata() y debería llenar nuestra variable (data) con los datos del archivo. Pero eso no funciona. La solucíon que encontré para eso, y que no es perfecta, es de dar solamente un vector (una dimensión) y de hacer una vuelta hasta llenar toda la tabla de 2 dimensiónes. Puede parecer un poco complicado así pero les invito a leer el código para entender lo que hice.

Así se termina esta parte sobre la lectura de los archivos HDF. Obviamente los algoritmos se puede mejorar, optimizar, pero creo que la apertura de un archivo es bastante rápida. En cambio, se podrÃa agregar la lectura de todos los tipos de archivos (VDatas). Les invito a consultar la documentacíon de la librería HDF : *http://www.hdfgroup.org/release4/doc/UsrGuide_html/UG_PDF.pdf*.

4. Visualización de imagénes

Esta parte no sera larga, querría solamente hablar de algunos problemas que encontré para mostrar un archivo SDS con la librería Qt. Y especialmente mostrarlos con los herramientas de la librería OpenCV para después ser capaz de modificar la imagen. Cuando abrimos un archivo SDS, como lo hemos visto, el archivo ExplorerWindow retorna una tabla de tipo uint16**. Aquí, ya encontramos un problema, en efecto la librería Qt y especialmente el Widget QGraphicScene soporta solamente las imágenes de 8 bits. Es un problema importante porque perdimos mucha información. En el futuro creo que sería interesante de poder abrirlas con 16 bits. O al menos hacer los tratamientos sobre una estructura Mat (de OpenCV) de 16 bits. Tal vez estarán sorprendido de ver que puse los datos del archivo SDS en una estructura Mat de 3 canales. En efecto, abrimos solamente una imagen en blanco y negro, no necesitamos estos 3 canales. Pero cuando vamos a ver los clasificadores de imagen, necesitaremos estos tres canales para mostrar si un pixel pertenece a una clase o a la otra con colores distintos.

5. Clasificadores de texturas

Llegamos finalmente a la parte la más importante de este informe : los clasificadores de texturas. Mi trabajo en esta parte fue de tratar los dos algoritmos LBP y GMRF. Empezaré explicando el principio de los dos y luego cuales son los problemas que encontré.

5.1. Local Binary Pattern

Este parágrafo se apoya principalmente sobre el articulo[1].

El principio del LBP

El principio del algoritmo LBP es muy facíl de entender. El objetivo cuando fue inventado este algoritmo fue de hacer un clasificador de textura independiente del contraste. Así que puede identificar una misma textura con dos exposiciónes al sol distintas. Consiste en calcular un indice. Abajo un ejemplo :

15	20	10
25	17	17
20	25	15

Figura 10: Ejemplo para calcular el LBP

Para calcular el indice LBP hacemos las diferencias entre el medio y todos los pixeles al rededor, y llenamos una matriz con la regla siguiente :

$$f(p_i - p_{medio}) = 1 \quad si\ p_i - p_{medio} >= 0 \tag{1}$$
$$= 0 \quad si\ p_i - p_{medio} < 0 \tag{2}$$

En el caso del ejemplo arribar tenemos :

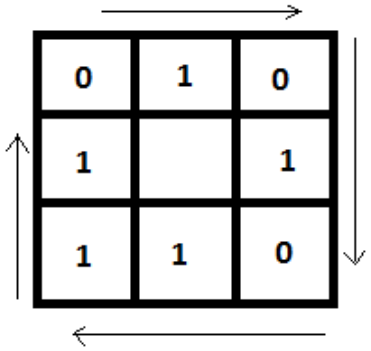

Figura 11: Matriz binaria

Así tenemos un numero binario en el sentido de la flecha. Veremos después porque el sentido es importante. En este caso nuestro numero es **01011110** y lo convertimos en decimal : 234. Este indice va a caracterizar este textura.
Por el momento si la textura esta en un otro lugar de la imagen pero girada de cualquier angulo, nuestro clasificador no va a "ver" que es la misma textura. Para que nuestro clasificador sea independiente de la rotacíon tenemos que "hacer" girar nuestro numero binario hasta encontrar el indice mínimo.

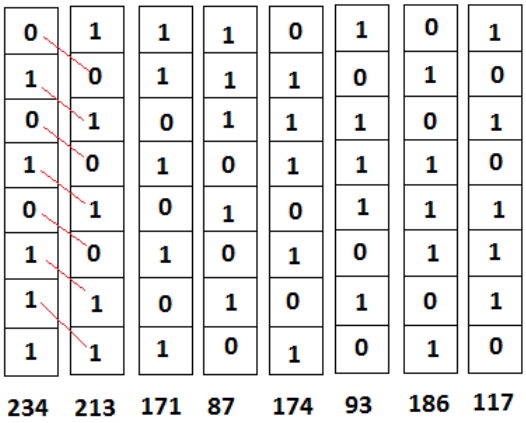

Figura 12: Rotacíon del vector

Entonces acá el indice que usaremos sera **87**. Si calculamos el LBP así el problema es que podemos clasificar una textura muy pequeña de 3 por 3 pixeles. Mi primero intento fue de calcularlo con una "ventana" de 5 por 5 pero así tenemos 2^{25} posibilidades de indice. Y cuando tendremos que calcular el histograma (ver depués) tendremos un vector de 2^{25} componentes. Un vector con tantas componentes es muy dificíl y pesado para manejar en C. Lo que propone el articulo [1] es calcular el LBP con los "pixeles" como lo muestra la figura 13.

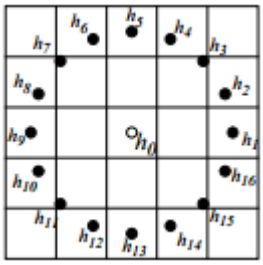

Figura 13: Otra manera de calcular el LBP (articulo [1])

El problema en este caso es que tenemos que calcular los valores de los "puntos" que no estan en el centro de un pixel por interpolacíon. Para explicar como hice este interpolacíon tomaré el ejemplo del punto h_4 en la figura 13. La idea es de tomar los valores de todos los pixeles al rededor de este punto (3 por 3) y de hacer un promedio donde los coeficientes son las distancias entre el punto y los centros de cada pixeles.

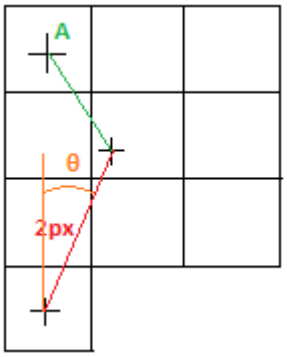

Figura 14: Interpolación

Primero calculamos las coordenadas de los 2 puntos (el punto h_4 y el punto A aquí).

$$X_A = 3 \tag{3}$$
$$X_A = 0 \tag{4}$$
$$y \tag{5}$$
$$X_{h_4} = 2\sin(\theta) \tag{6}$$
$$Y_{h_4} = 2\cos(\theta) \tag{7}$$

Donde θ quí es igual a 22,5 grados.
Y sencillamente tenemos la distancia con la formula :

$$d = \sqrt{(X_A - X_{h_4})^2 + (Y_A - Y_{h_4})^2}$$

Y eso proceso lo hacemos para todos los pixeles al rededor del punto. Para mas detalles de como lo implementé en el programa se puede ver el algoritmo en el archivo ProcessingFunctions.cpp.

La primera etapa del clasificador es de decir al algoritmo : "Esos pixeles corresponden a la clase 1, esos otros corresponden a la clase 2" como lo muestra la figura [15].

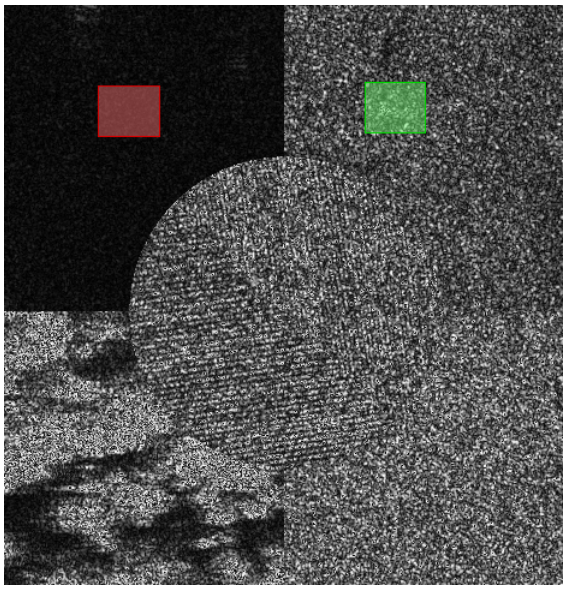

Figura 15: Elegir las clases

La pregunta es : "Que hacemos con estos pixeles ?" Calculamos el LBP para cada pixel y con estos indice construimos un histograma que va de 0 a 2^{16}. La figura 16 muestra los histogramas de los 2 clases elegidas en la figura 15.

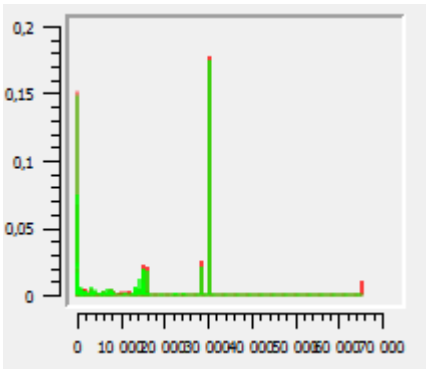

Figura 16: Elegir las clases

Ahora mismo podemos ver que tenemos un problema importante : los dos histogramas son casi idénticos. Entonces la clasificacíon no sera buena. Hablaremos de los problemas más con detalles luego. Después haber elegido las clases tenemos que decir al algoritmo : "quiero analizar estos pixeles". Lo que significa eligir una ventana de pixeles donde queremos saber si pertenecen a la clase 1 o la 2. Entonces para cada pixeles tomamos una ventana de 7 por 7 donde calculamos los indices de todos los pixeles. Eso nos da un histograma que podemos comparar con los histogramas de las 2 clases elegidas. El histograma que se acerca lo mas del histograma calculado corresponde a la clase del pixel.
El algoritmo va calculando y luego nos muestra la pertenencia de cada pixel con colores (Ver figura 17).

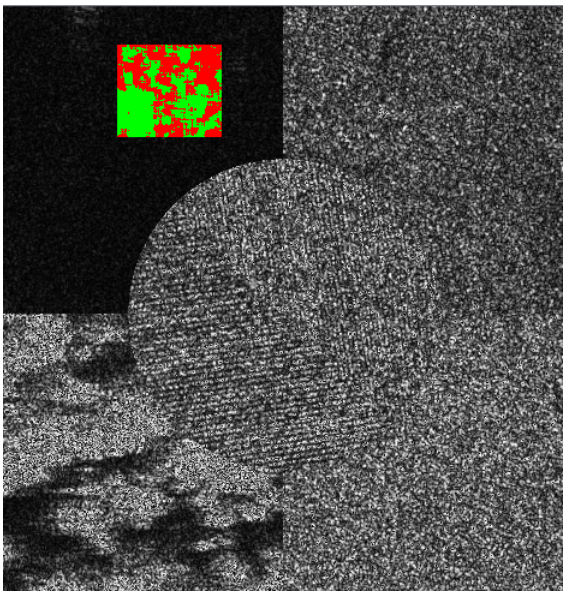

Figura 17: Elegir las clases

Vemos claramente con esta imagen que el clasificador no funciona.

Los problemas del LBP

Vimos que el algoritmo implementado en el programa no funciona. Lo que tenemos que saber es "Porque ?".

- Primero creo que hay un problema de calculo cuando elegimos las clases. En efecto como lo vimos los histogramas son casi iguales. Con esta condición es imposible que el clasificador hace su trabajo correctamente.
- Segundo como lo diremos antes el LBP es independiente del contraste. El problema es que perdemos mucha información así. El articulo [1] propone una solución para implementar la varianza de los valores de los pixeles. Creo que es imprescindible de implementar esta solución para que el LBP sea interesante en nuestro caso.

5.2. Gaussian Markov Random Field

El principio

La idea de hacer un clasificador así viene de la constatación que los pixeles que son vecinos tiene una relación entre ellos, se influencien recíprocamente. La hipótesis que es hecha en el caso de un GMRF es que en una ventana de pixeles los valores tienen una repartición gaussiana. En esta condicíon podemos calcular el promedio y la varianza de los valores dentro una ventana elegida. Tomamos el ejemplo de una ventana de 3 por 3 : podemos calcular el promedio y la varianza así :

$$\mu = \frac{1}{9}\sum_{n=0}^{9} p_n \tag{8}$$

$$Var = \frac{1}{9}\sum_{n=0}^{9} (p_n - \mu)^2 \tag{9}$$

Entonces cuando elegimos las clases un algoritmo va calculando el promedio y la varianza de la ventana de pixeles elegida. Luego cuando hacemos la clasificación de pixeles calculamos el promedio y la varianza de una ventana de 3 por 3 al rededor del pixel que queremos clasificar. Depués tenemos que comparar el promedio y la varianza con los de las clases elegidas. Para eso usamos un clasificador Bayesiano.
Según el teorema de Bayes tenemos :

Teorema de Bayes

$$P(\omega|f) = \frac{P(f|\omega)P(\omega)}{P(f)} \tag{10}$$

(11)

Donde ω es una clase y f la medida por cual queremos clasificar.

Dado que P(f) estara constante, no importa para la clasificación. Necesitamos entonces $P(f|\omega)$ y $P(f)$. Ya hicemos la hipótesis que los valores se distribuyen de manera gaussiana. Entonces :

$$P(f|\omega) = \prod_{c\in C} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(f-\mu)^2}{2\sigma^2}}$$

Formula que corresponde a la función de Gauss. $P(\omega)$ es calculado en función de los vecinos del pixel central con la formula

$$P(\omega) = \frac{1}{Z} e^{-U(\omega_s)}$$

donde Z es un a constante de normalización, ω_s es la clase con quien comparamos el pixel y U se llama energía y se calcula con la formula :

$$U(\omega_s) = \sum_{c\in C} V_c(\omega_s)$$

donde C es el conjunto de los vecinos del pixel central y $V_c(\omega)$ se calcula :

$$V_c(\omega) = \beta\delta(\omega_i, \omega_j)$$

Si tomamos de nuevo el teorema de Bayes tenemos :

$$P(\omega|f) = P(f|\omega)P(\omega) \tag{12}$$

$$= \prod_{c\in C} \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(f-\mu)^2}{2\sigma^2}} \frac{1}{Z} e^{-U(\omega_s)} \tag{13}$$

$$-log(P(\omega|f)) = \sum_{c\in C} \left[\frac{1}{2} log(2\pi\sigma^2 Z^2) + \frac{(f-\mu)^2}{2\sigma^2}\right] + U(\omega_s) \tag{14}$$

con $U(\omega_s) = \sum_{c\in C} V_c(\omega_s)$ y $U(\omega_s|f) = -log(P(\omega_s|f)$ tenemos finalmente :

Resultado

$$U(\omega|f) = \sum_{c\in C} \left[\frac{1}{2} log(2\pi\sigma^2 Z^2) + \frac{(f-\mu)^2}{2\sigma^2}\right] + \sum_{c\in C} \beta\delta(\omega_s, \omega_c) \tag{15}$$

Queremos maximizar la probabilidad que sea de la clase ω dado f. Entonces queremos minimizar $U(\omega|f)$. Todo eso es la parte matemática del problema. Ahora vamos a ver como lo he implementado en el programa.

- Selección de las clases :
  Cuando el usuario va a seleccionar una zona en la imagen el algoritmo va solamente calculando el promedio y la varianza empírica de esta zona.
- Clasificación :
  La primera etapa es de calcular el promedio y la varianza asociados a cada pixel de la zona donde queremos saber la pertenencia de los pixeles a una clase. El promedio y la varianza son calculados con los vecinos del pixel central (puede ser una zona de 3x3;5x5;7x7;9x9). La segunda etapa es de calcular el $-log(P(f|\omega))$ por cada clase y cada pixel y de eligir el omega (la clase) qui minimiza este logaritmo. Y la tercera y ultima etapa es de calcular el $U(\omega|f)$ y eligir la clase qui lo minimiza. Finalemente cada pixel sera clasificado.

Los resultados

El algoritmo funciona bastante bien sobre las imágenes satélites como lo muestra la figura 18 :

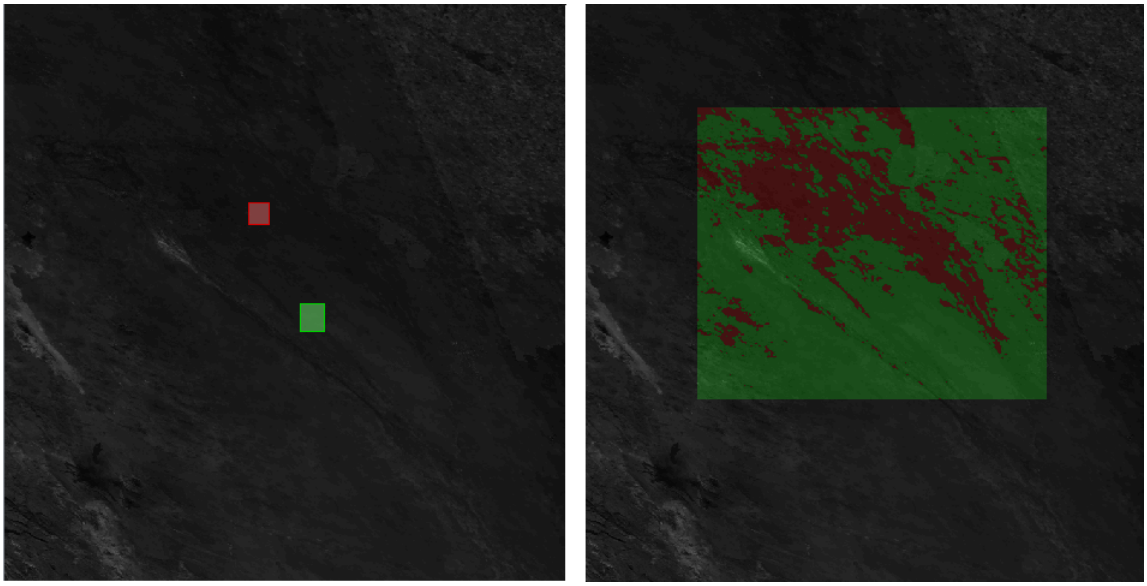

Figura 18: Resultado GMRF sobre una imagen satélite

En cambio funciona muy mal sobre la imagen de prueba siguiente :

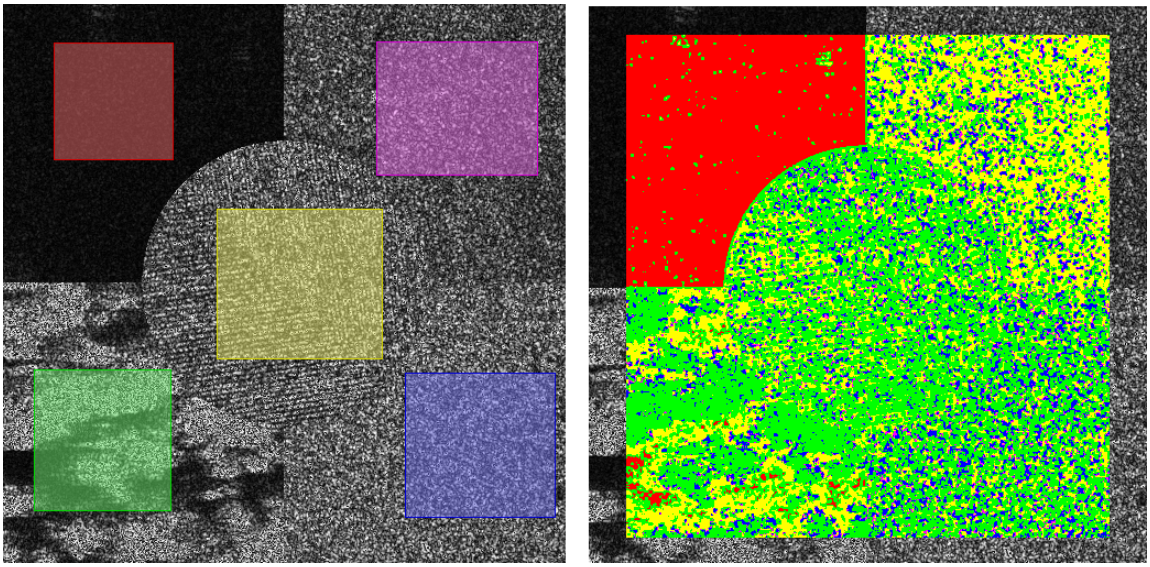

Figura 19: Resultado GMRF sobre la imagen de prueba

Los problemas

Appendix: Robot Design and Engineering