SISTEMA PARA LA OBTENCIÓN DE UN ELASTOGRAMA DE TEJIDOS BIOLÓGICOS

Anexos complementarios

Cristian Felipe Gutiérrez Pazos
Byron Yesid Jaramillo Hurtado

Director: Mg. Elena Muñoz España

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Electrónica, Instrumentación y Control
Ingeniería en Automática Industrial
Popayán, Abril de 2015

TABLA DE CONTENIDO

LISTA DE FIGURAS

Anexo A: Manual de instalación del programador

El objetivo de este manual, es permitir a los usuarios realizar mejoras y/o cambios al desarrollo hecho dentro del software de elastografía. Para ello, se explicara el paso a paso de la instalación de OpenCV, Qt y Matlab Compiler Runtime, y poder hacer uso de estas bibliotecas sobre Visual Studio.

A.1. Instalación de OpenCV 2.4.X.

Para este caso de estudio, fue utilizada la versión 2.4.6. Si desea, puede descargar la versión más reciente en el siguiente enlace: OpenCV 2.4.X.

1. Crear una carpeta llamada “OpenCV”. Se recomienda que esta carpeta sea creada en el disco “C” de la siguiente manera:

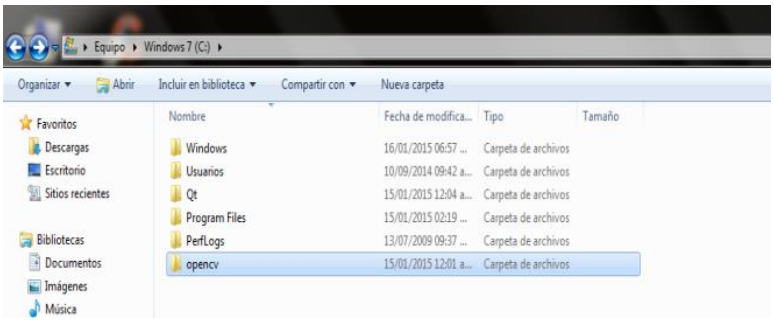

Figura A. 1. Creación de directorio para OpenCV

2. Dentro de la carpeta creada anteriormente, descomprimir los archivos que se descargaron. Deberán quedar de la siguiente manera:

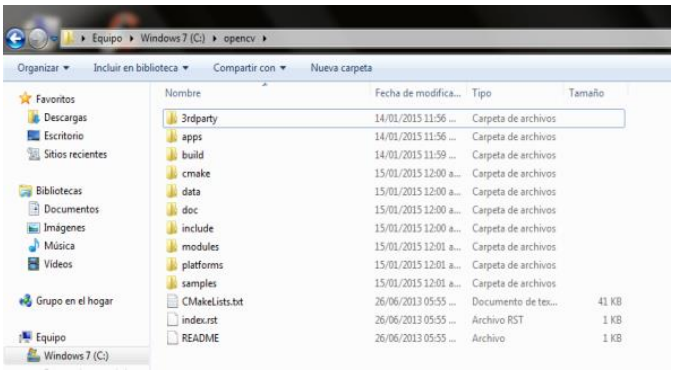

Figura A. 2. Carpeta de archivos OpenCV

Ahora se procede a configurar las variables de entrono, para habilitar el uso de OpenCV en el equipo.

3. Sobre el icono de “Equipo” dar clic derecho, en el menú de opciones que despliega dar clic en “Propiedades”. Aparecerá una ventana como esta:

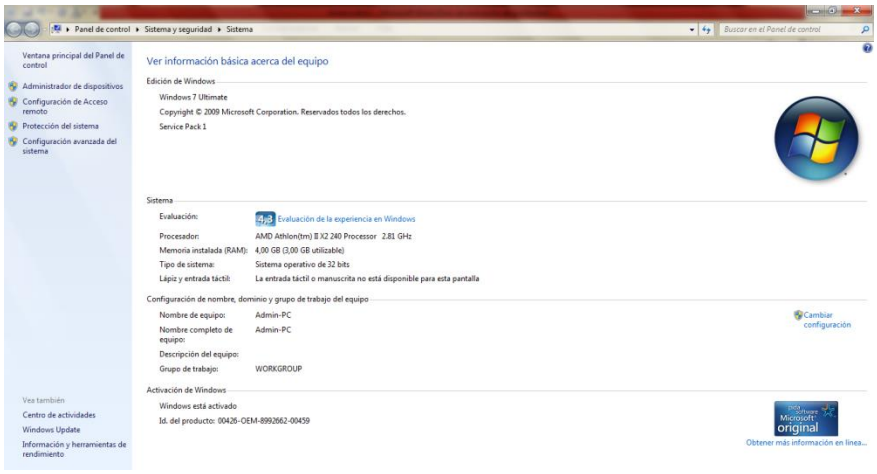

Figura A. 3. Propiedades de Equipo

De aquí se puede resaltar el tipo de sistema, para este caso es un sistema de **32 bits**. Por lo tanto el tipo de arquitectura es **x86**.

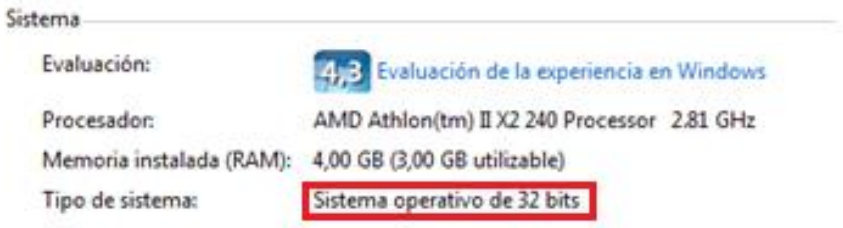

Figura A. 4. Tipo de sistema operativo

Es importante saber el tipo de arquitectura, ya que así se puede seleccionar el tipo de compilador a utilizar, en este caso es **vc11.** Si se desea saber qué tipos de compilador soporta, se puede ir a la dirección **"C:/opencv/build/x86"**.

4. Ahora se configuran las variables del entorno, en la misma ventana de propiedades, en la parte superior izquierda, dar clic en "Configuración Avanzada del Sistema".

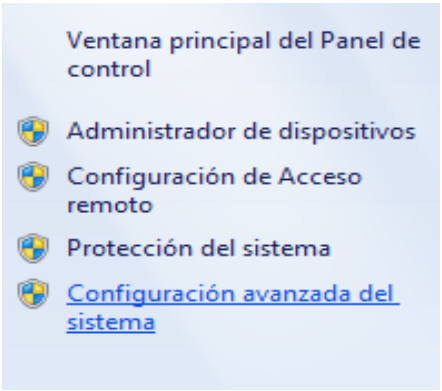

Figura A. 5. Opción de configuración avanzada del sistema

5. Sobre la nueva ventana, dar clic a la opción "Variables de entorno…".

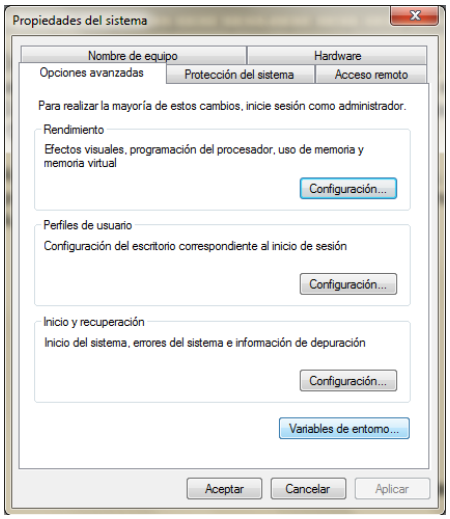

Figura A. 6. Ventana de configuración propiedades del sistema

6. Sobre la ventana de “Variables de entorno”, dar clic al botón “Nueva…” en la parte inferior de esta.

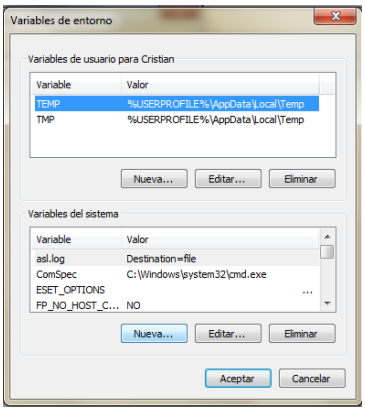

Figura A. 7. Ventana de configuración de variables de entorno

Ahora se procede a configurar el valor de la nueva variable con la siguiente información:

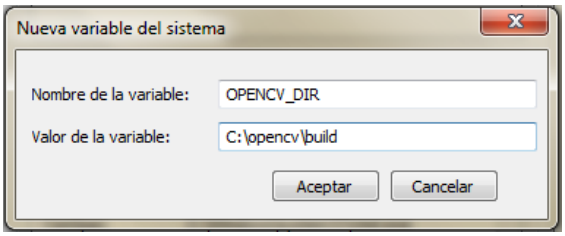

Figura A. 8. Configuración de nueva variable de entorno

Dar clic en aceptar. Con esto se ha terminado de agregar una nueva variable de entorno al sistema.

Ahora se procede a editar la variable del sistema.

7. En la ventana de “Variables de entorno”, en la parte inferior se encuentra una ventana llamada “Variables del sistema”, seleccionar la línea **‘Path’** y dar clic al botón “Editar…”.

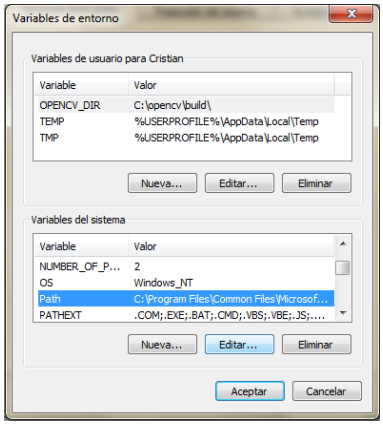

Figura A. 9. Configuración variable del sistema.

8. En la nueva venta se encuentra una lista de direcciones pre-configuradas, no borrar ninguna, solo agregar **‘;’** a la línea final y adicionar la siguiente dirección **‘%OPENCV_DIR%\x86\vc11\bin’**.

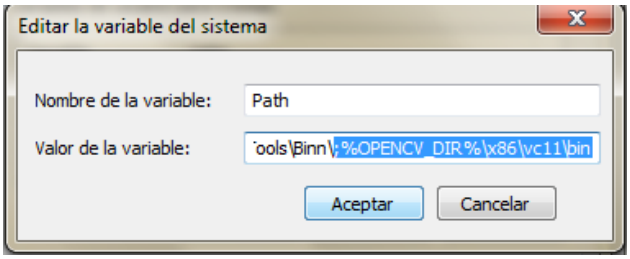

Figura A. 10. Adición de la variable del sistema para OpenCV

Como se mencionó anteriormente, es necesario saber el tipo de arquitectura y el tipo de compilador a usar (**Paso 3**), ya que con estos datos se configurara la variable del sistema.

Con esto, se finaliza la instalación de OpenCV en el computador, ahora solo queda por configurar su uso en Visual Studio.

A.2. Configuración de OpenCV en Visual Studio.

Para completar la guía de instalación de OpenCV, es necesario configurar su uso en Visual Studio. Para ello solo se debe crear una hoja de propiedades en este proyecto.

1. Abrir el proyecto o solución, o crear un nuevo proyecto si está empezando. Una vez abierto o creado, deberá verse de la siguiente manera.

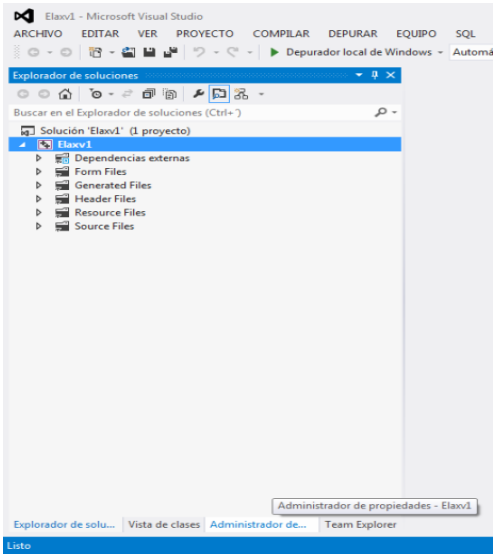

Figura A. 11. Explorador de soluciones Visual Studio

2. En la parte inferior del explorador de soluciones, dar clic sobre la pestaña **'Administrador de propiedades'**.

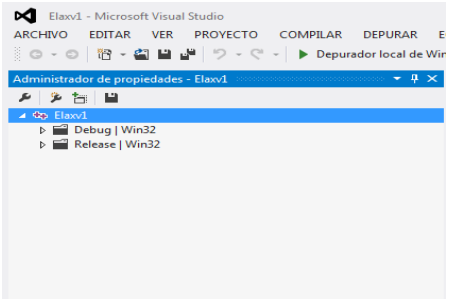

Figura A. 12. Administrador de propiedades Visual Studio

En esta ventana se encuentran dos opciones. Estas opciones dependen de que configuración que se hayan escogido para compilar la solución, ya sea **'Debug'** o **'Release'.**

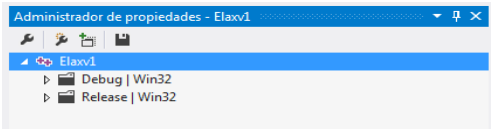

Figura A. 13. Modos de configuración administrador de propiedades

Para este caso de estudio, se configuran las propiedades para el modo **'Release'.**

3. Dar clic derecho sobre la opción **'Release|Win32'**. Seleccionar la opción **'Agregar nueva hoja de propiedades al proyecto…'**.

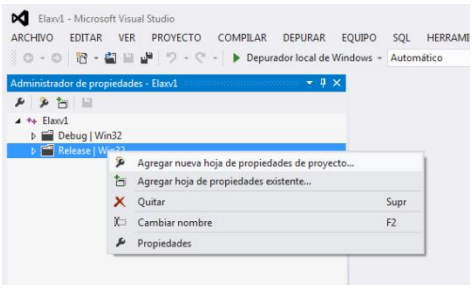

Figura A. 14. Hoja de propiedades

Aparecerá una pantalla como la que muestra la figura B.5, donde se configura el nombre y la ubicación de la nueva hoja de propiedades. Se recomienda dejar la ubicación que dan por defecto.

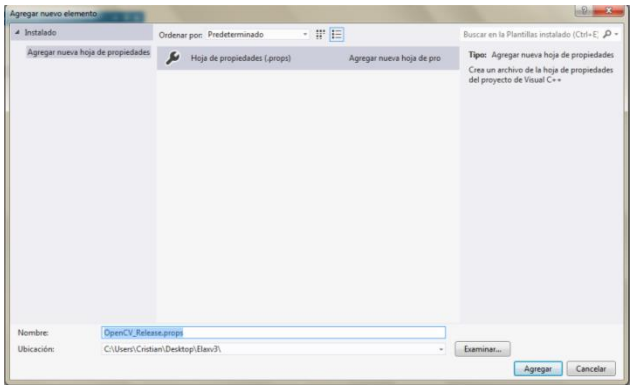

Figura A. 15. Configuración nueva hoja de propiedades.

En este ejemplo la nueva hoja de propiedades se llama **'OpenCV_Release'**. Dar clic en "Agregar". Se despliega la opción **'Release|Win32'** en el administrador de propiedades y se puede observar la hoja de propiedades creada.

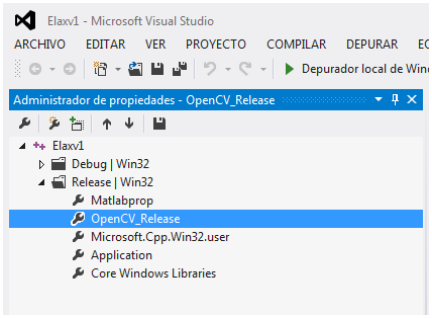

Figura A. 16. Hoja de propiedades OpenCV_Release

4. Dar clic derecho sobre la hoja de propiedades creada, y seleccionar la pestaña **'Propiedades'**. Aparecerá una ventana como la siguiente.

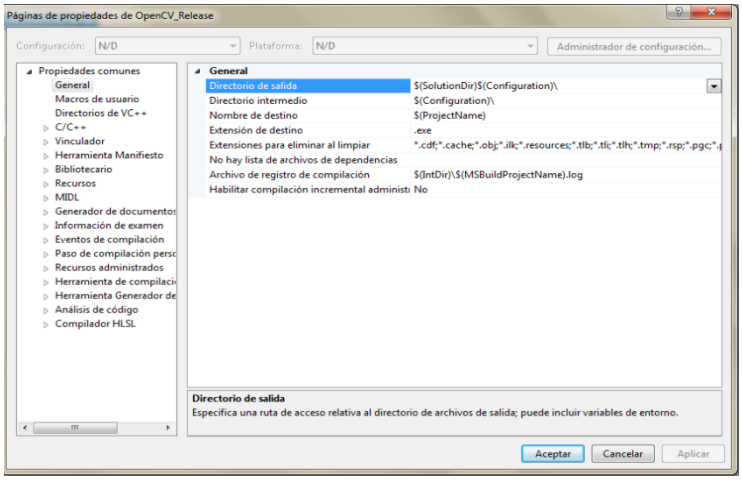

Figura A. 17. Propiedades de la hoja OpenCV_Release

5. Desplegar el sub-directorio **'C/C++'**, seleccionar la opción **'General'**. En la parte derecha de la ventana, modificar la opción **'Directorios de inclusión adicionales'**. En esta pestaña, colocar la dirección donde se ubican los archivos "include" de OpenCV: **'$(OPENCV_DIR)\include'**.

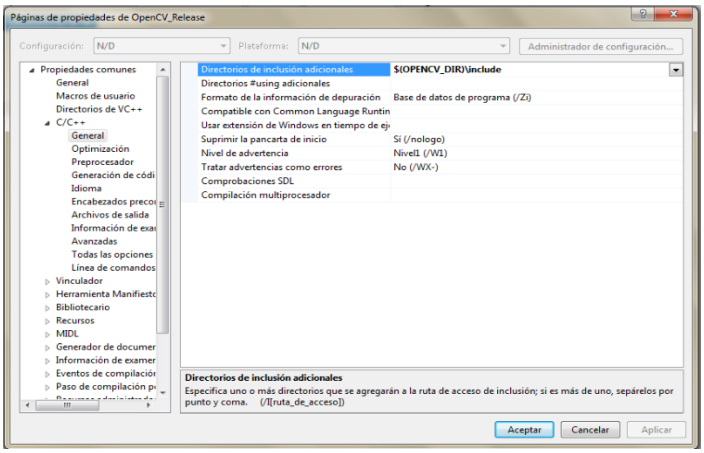

Figura A. 18. Configuración directorios de inclusión OpenCV

Dejar las demás opciones configuradas por defecto.

6. Desplegar el sub-directorio **'Vinculador'**, seleccionar la opción **'General'**. En la parte derecha de la ventana, modificar la opción **'Directorios de bibliotecas adicionales'**. En esta pestaña, poner la dirección donde se ubican los archivos "lib" de OpenCV: **'$(OPENCV_DIR)\ x86\vc11\lib'**.

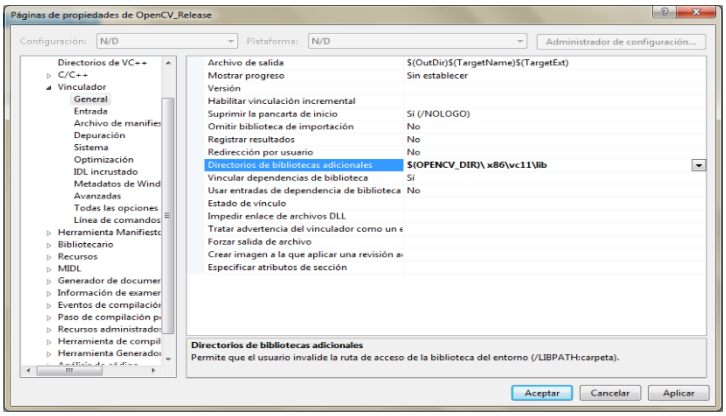

Figura A. 19. Configuración directorio de archivos 'lib' OpenCV

Dejar las demás opciones configuradas por defecto.

7. En el mismo sub-directorio **'Vinculador'**, seleccionar la opción **'Entrada'**. En la parte derecha de la ventana, modificar la opción **'Dependencias adicionales'**. Dar clic sobre la pestaña y seleccionar la opción **'<Editar…>'**.

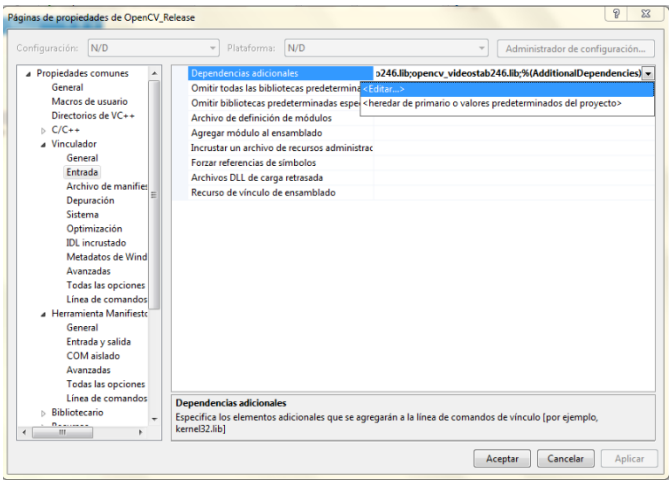

Figura A. 20. Configuración y edición de dependencias adicionales

Después de haber dado clic en la opción **'<Editar…>'**, apareceré un cuadro de dialogo. En ese cuadro se pondrán los nombres de los archivos de librería de OpenCV que se a usaran.

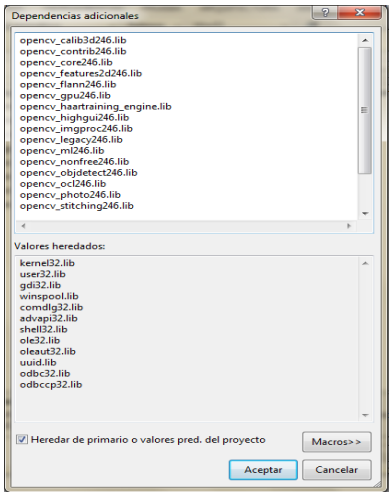

Figura A. 21. Adición archivos de librerías OpenCV

Los nombres de estor archivos, se pueden encontrar en la ubicación **'C:\opencv\build\x86\vc11'**.

"opencv_calib3d246.lib
opencv_contrib246.lib
opencv_core246.lib

opencv_features2d246.lib
opencv_flann246.lib
opencv_gpu246.lib
opencv_haartraining_engine.lib
opencv_highgui246.lib
opencv_imgproc246.lib
opencv_legacy246.lib
opencv_ml246.lib
opencv_nonfree246.lib
opencv_objdetect246.lib
opencv_ocl246.lib
opencv_photo246.lib
opencv_stitching246.lib
opencv_superres246.lib
opencv_ts246.lib
opencv_video246.lib
opencv_videostab246.lib”

Dar clic en el botón “aceptar”.

Con esto se termina la configuración e instalación de OpenCV en el equipo. La hoja de propiedades creada es portable, por tanto se puede usarla en otros proyectos, copiando el archivo en la carpeta raíz del nuevo proyecto.

Para configurar la hoja de propiedades para el modo **‘Debug’**, realizar los mismos pasos, pero al llegar al paso 7, las librerías que se agregan son las siguientes:

“opencv_calib3d246d.lib
opencv_contrib246d.lib
opencv_core246d.lib
opencv_features2d246d.lib
opencv_flann246d.lib
opencv_gpu246d.lib
opencv_haartraining_engined.lib
opencv_highgui246d.lib
opencv_imgproc246d.lib
opencv_legacy246d.lib
opencv_ml246d.lib
opencv_nonfree246d.lib
opencv_objdetect246d.lib
opencv_ocl246d.lib
opencv_photo246d.lib
opencv_stitching246d.lib

opencv_superres246d.lib
opencv_ts246d.lib
opencv_video246d.lib
opencv_videostab246d.lib”

A.3. Instalación de Qt v5.x.x

Para este caso de estudio, es utilizada la versión 5.1.1. Si desea, puede descargar la versión más reciente en el siguiente enlace: Qt v5.x.x.

Adicionalmente, descargar un programa llamado **‘Visual Studio Add in 1.2.2 for Qt5’**, el cual permitirá crear automáticamente las variables de entorno para Qt y creara un acceso directo, en la pestaña de herramientas de Visual Studio, y así poder crear proyectos sin salir del IDE. El programa se puede descargar del enlace mencionado al principio de este punto.

1. Instalar el archivo .exe que se descargo de la página**.**

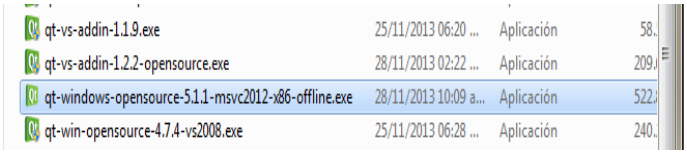

Figura A. 22. Archivo de instalación Qt 5.1.1

2. Dar clic en **‘Next’**, en la ventana que se abre.

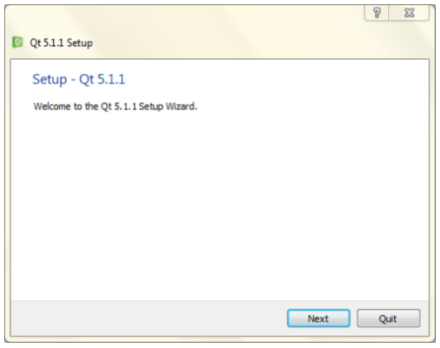

Figura A. 23. Ventana de inicio instalación Qt.

3. En esta ventana se especifica la ubicación, donde se instalará Qt, Utilizando la que se tiene por defecto. Dar clic en **‘Next’.**

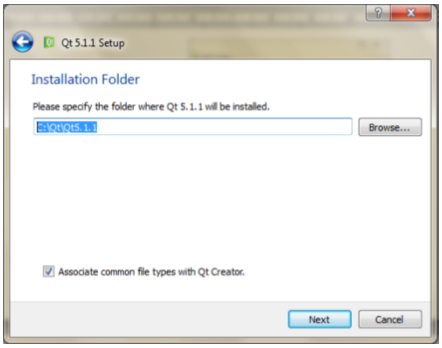

Figura A. 24. Configuración carpeta de instalación Qt.

4. La ventana siguiente permite seleccionar los componentes que se desean en la instalación. Por defecto Qt selecciona los componentes necesarios para su uso. Dar clic en **'Next'.**

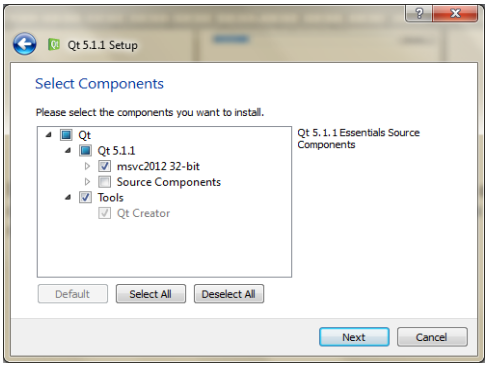

Figura A. 25. Selección de componentes de instalación Qt

5. En esta ventana se muestra los acuerdos de licencia para uso de Qt. Aquí se debe seleccionar la opción que dice "I have read and agree to the following…", el cual habilitará la opción para continuar la instalación. Dar clic en **'Next'.**

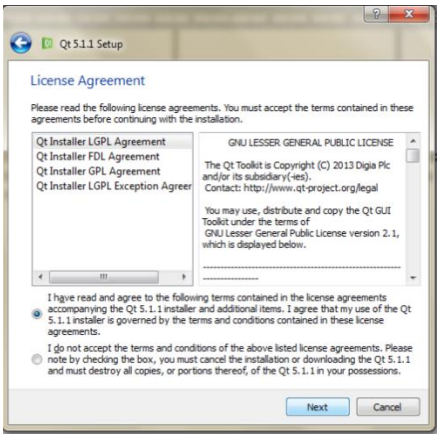

Figura A. 26. Licencias y acuerdos de uso de Qt

6. Crear los accesos directos en el menú de inicio, con el nombre que dan por defecto. Damos clic en **'Next'.**

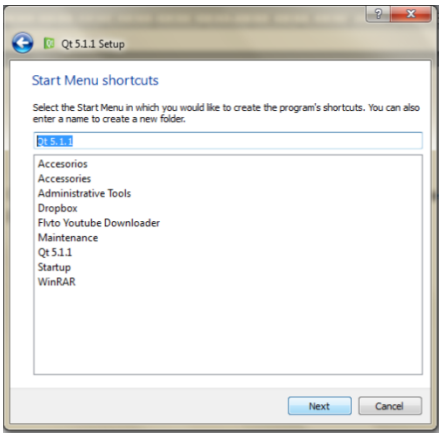

Figura A. 27. Creación de acceso directo

7. Dar clic en el botón **'Install'**, para proceder a instalar Qt en el computador.

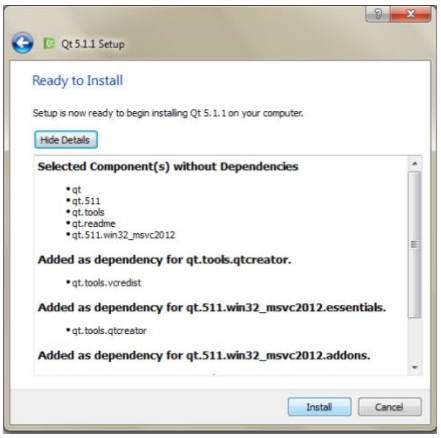

Figura A. 28. Instalación de componentes seleccionados

8. Terminada la instalación, dar clic en el botón **'Finish'**.

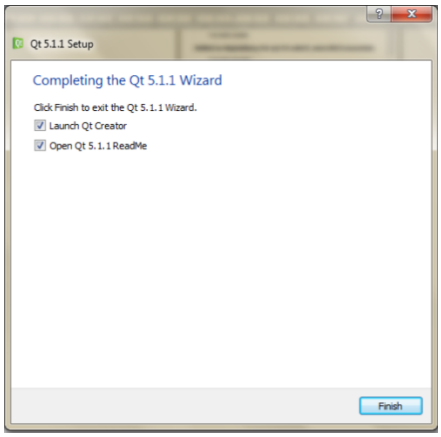

Figura A. 29. Finalización de la instalación de Qt

Con esto esta terminada la instalación de Qt. Ahora se procede a instalar el Add-in que permitirá el uso de Qt sobre Visual Studio.

9. Instalar el archivo .exe, que se descarga de la página.

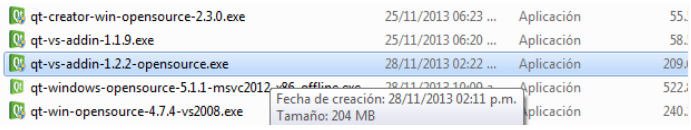

Figura A. 30. Archivo de instalación Qt Add-in

10. Dar clic en **'Next'**, en la ventana que se abre.

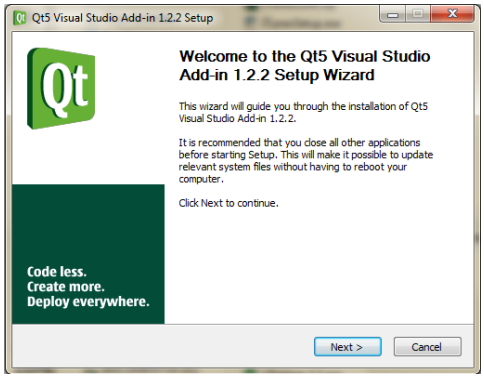

Figura A. 31. Ventana de inicio instalación Qt Add-in

11. En esta ventana se muestra los acuerdos de licencia para uso de Qt add-in. Aquí, seleccionar la opción que dice "I accept the terms of the license…", el cual habilitará la opción para continuar la instalación. Dar clic en **'Next'.**

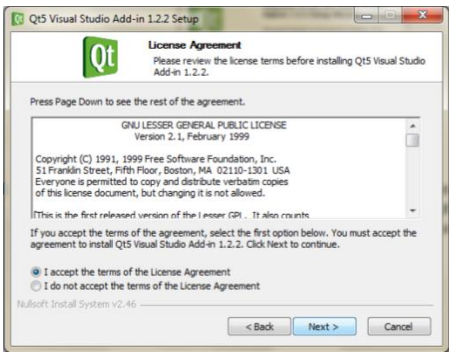

Figura A. 32. Licencia y acuerdo de uso Qt Add-in

12. Dejar las opciones que se dan por defecto y dar clic en **'Next'**, en la ventana que se abre.

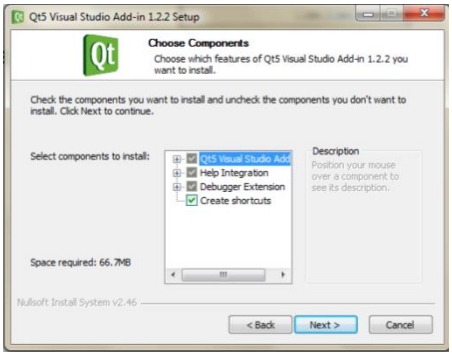

Figura A. 33. Selección de componentes de instalación Qt Add-in

13. Dar clic en **'Install'**, para que proceda a instalar la herramienta.

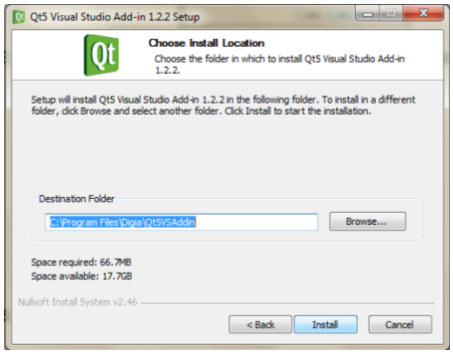

Figura A. 34. Instalación de componentes seleccionados

14. Finalizada la instalación, dar clic en **'Next'** y por último en **'Finish'**.

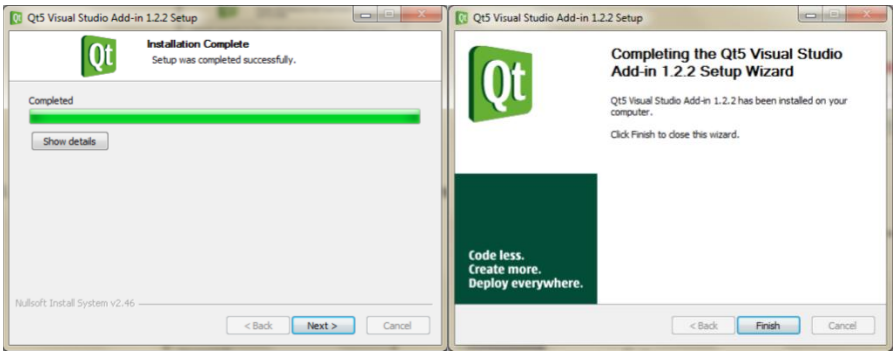

Figura A. 35. Finalización instalación Qt Add-in

Una vez finalizada la instalación del Add-in de Qt, se procede a abrir Visual Studio para configurar el uso de la herramienta sobre este IDE.

15. Abrir Visual Studio, en la “Barra de herramientas” se ha instalado una nueva pestaña que permite la configuración y el uso de Qt en Visual Studio como lo muestra la figura A.36.

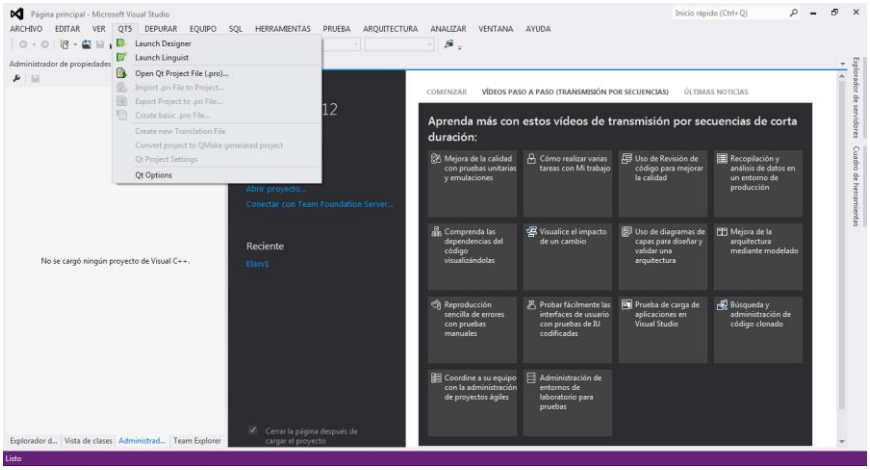

Figura A. 36. Pestaña de configuración Qt en Visual Studio

16. Dar clic a la opción **‘Qt options’** en la pestaña de configuración. En esta opción se configura el directorio donde se instaló la herramienta Qt.

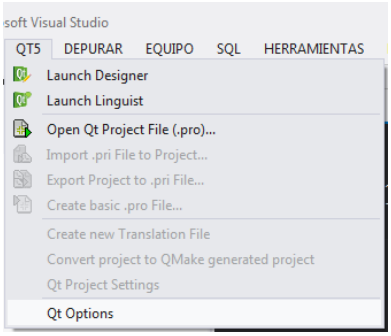

Figura A. 37. Opciones de Qt en Visual Studio

17. En la nueva ventana llamada **‘Qt options’**, dar clic en el botón **‘Add’**, se abrirá una nueva ventana donde se especificara la ubicación y la versión de Qt que se instaló con anterioridad. Se configurara la ventana con la información que se muestra en la figura C.18.

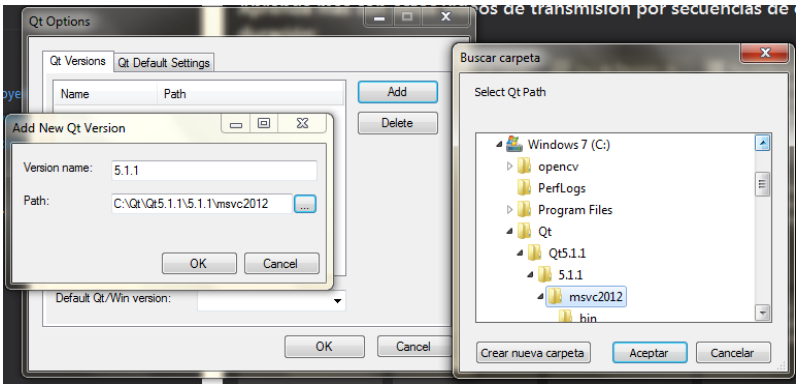

Figura A. 38. Configuración del Path de Qt para Visual Studio

En la casilla “Version name” pondremos el nombre de la versión de Qt que se vaya a utilizar, para este caso se pondrá **‘5.1.1’**. En la casilla “Path” se ubicara la siguiente dirección **‘C:\Qt\Qt5.1.1\5.1.1\msvc2012’**. En esta carpeta se encuentran los archivos necesarios para que Qt funcione sobre Visual Studio.

18. Una vez terminado este proceso, dar clic en **‘ok’**.

Con esto se termina la instalación y configuración de Qt en el equipo.

A.4. Instalación y configuración de Matlab Compiler Runtime

Para completar la guía de instalación para el programador, se explica el proceso de instalación de la herramienta **‘Matlab Compiler Runtime 2013b 32 bits’**. Esta herramienta, es un conjunto independiente de librerías compartidas que habilitan la ejecución de aplicaciones creadas en Matlab, en un entorno basado en C++, sin necesidad de instalar o tener instalado Matlab. Cabe resaltar que si tiene Matlab instalado en su ordenador, no es necesaria la instalación de esta herramienta.

Para este caso de estudio, se utiliza la versión del MCR 2013a. Si desea, puede descargar la versión más reciente en el siguiente enlace: MCR 20XX.

1. Dar doble clic sobre el archivo descargado de la página de Matlab.

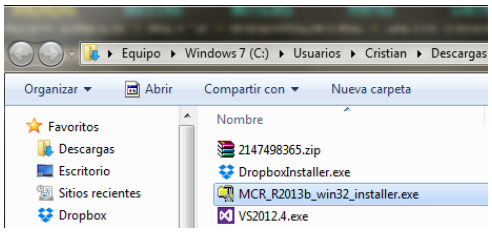

Figura A. 39. Archivo de instalación MCR 2013b

2. Después de descomprimir los archivos necesarios para la instalación, aparecerá una ventana nueva. Dar clic en **'Next >'**.

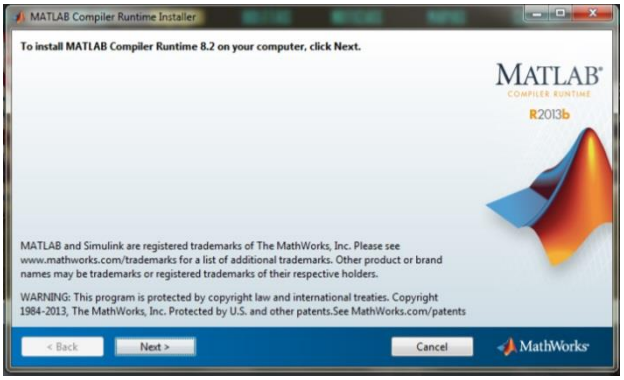

Figura A. 40. Ventana de inicio instalación MCR 2013b

3. Seleccionar la palabra **'yes'**, para aceptar los términos de licencia y uso de MCR en el ordenador. Dar clic en **'Next >'**.

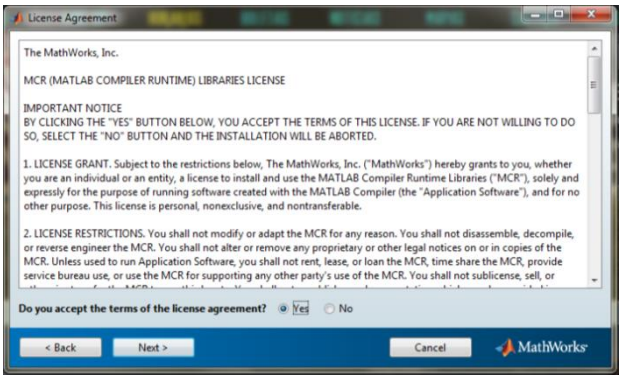

Figura A. 41. Ventana acuerdos de licencia y de uso MCR 2013b

4. Seleccionar la ubicación donde se desea instalar las librerías. Dejar por defecto la ubicación dada. Dar clic en **'Next >'**.

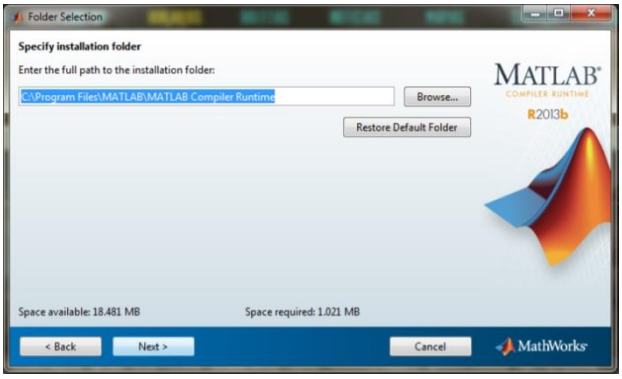

Figura A. 42. Ventana selección folder de instalación MCR 2013b

5. En la nueva ventana, dar clic en **'Install >'**. Empezara el proceso de instalación del MCR.

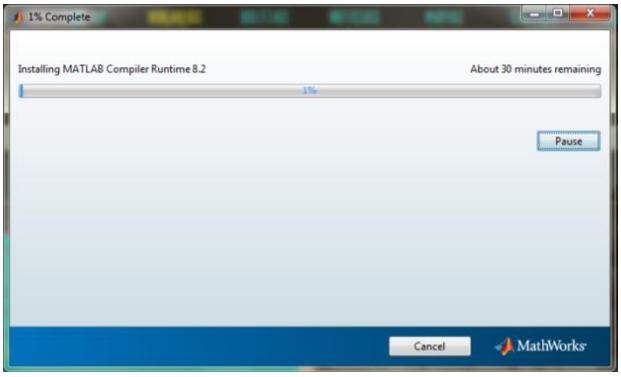

Figura A. 43. Ventana proceso de instalación MCR 2013b

6. Para terminar la instalación dar clic en **'Finish'**.

7. Una vez terminado el proceso de instalación, se procede a configurar su uso para Visual Studio. Para esto se repiten los cuatro primeros pasos mencionados en el apéndice **'B.1. Configuración de OpenCV en Visual Studio'**. El nombre que se utiliza para la hoja de propiedades es 'Matlabprop'.

En la hoja de propiedades Matlabprop seguir los siguientes pasos.

1. Desplegar el sub-directorio **'Vinculador'**, seleccionar la opción **'General'**. En la parte derecha de la ventana, modificar la opción **'Directorios de bibliotecas adicionales'**. En esta pestaña, indicar la dirección donde se ubican los archivos "lib" de OpenCV: **'C:\Program Files\MATLAB\MATLAB_Compiler_Runtime\v81\extern\lib\win32\microsoft'.** Adicionalmente se agregará el directorio donde se encuentran los archivos de la "librería compartida" que se creó con anterioridad (para más información leer Anexo C).

   Dejar las demás opciones configuradas por defecto.

2. En el mismo sub-directorio **'Vinculador'**, seleccionar la opción **'Entrada'**. En la parte derecha de la ventana, modificar la opción **'Dependencias adicionales'**. Dar clic sobre la pestaña y seleccionar la opción **'<Editar…>'**.
   Después de haber dado clic en la opción **'<Editar…>'**, aparecerá un cuadro de dialogo. En ese cuadro se pondrán los nombres de los archivos de librería de Matlab a usar.

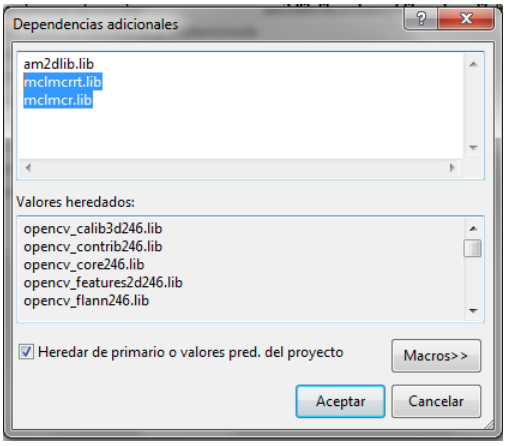

Figura A. 44. Ventana de configuración librerías de matlab en Visual Studio

Los archivos que se incluyen son **mclmcrrt.lib** y **mclmcr.lib**, los cuales permitirán el uso del Runtime de matlab. Adicionalmente se incluyeron los archivos creados en las librerías compartidas para uso en Visual Studio.

Con esto se termina la instalación y configuración del Compiler Runtime de Matlab para Visual Studio.

Anexo B: Creación de recursos e interfaces [50]

Este anexo fue tomado de [50], aquí se explicara la creación de ventanas y recursos que se usaran en la interfaz, de una manera práctica y reusable para la creación de proyectos bajo esta herramienta.

Esta es una de las tareas iniciales o previas a empezar a desarrollar un proyecto, no es necesario tener todas las imágenes para recursos elegidas, pero es importante configurar primero esto ya que en fases posteriores del proyecto sería realmente problemático hacer esto desde visual.

B.1. Creación de recursos.

1. Dentro de la carpeta con el código fuente se crea una carpeta para almacenar los recursos a usar. Por recursos hace referencia a todas las imágenes e iconos que se usaran para desarrollar el software, así:

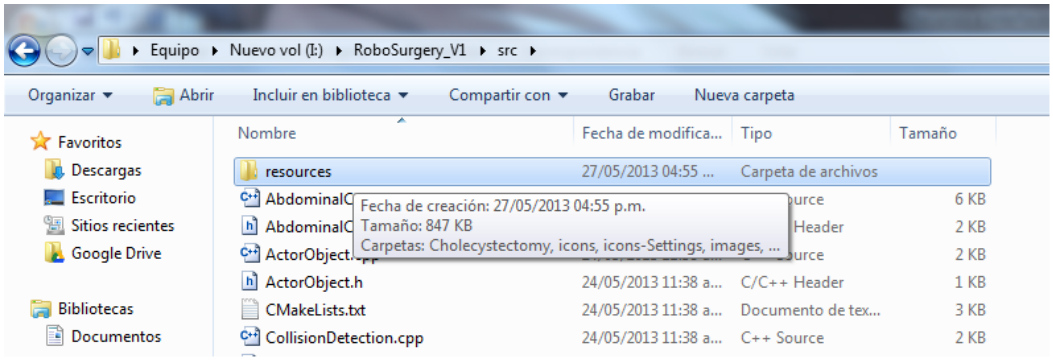

Figura B. 1. Creando carpeta de recursos dentro de la carpeta "src".

2. Dentro de la misma se guardan los archivos a utilizar, preferiblemente en formato '.png' ya que este permite guardar la opacidad de una imagen y así quitarle el fondo blanco (usando un programa de apoyo como por ejemplo GIMP).

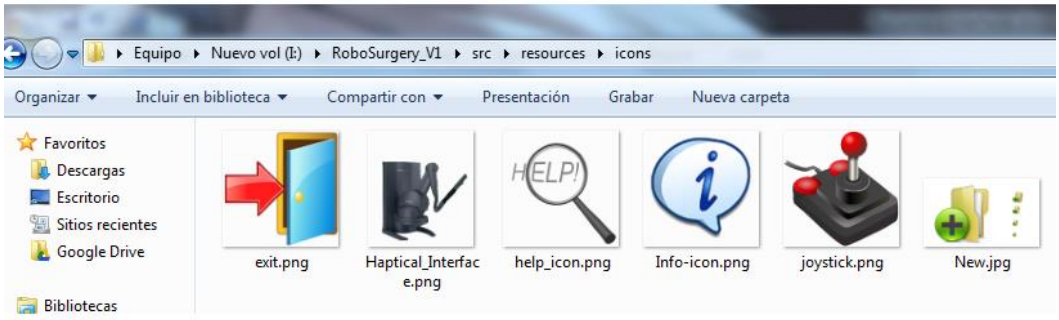

Figura B. 2. Guardando recursos para usar en el proyecto.

3. Abrir (o crear) el proyecto en Qt-Creator, clic derecho en el proyecto, y agregar nuevo:

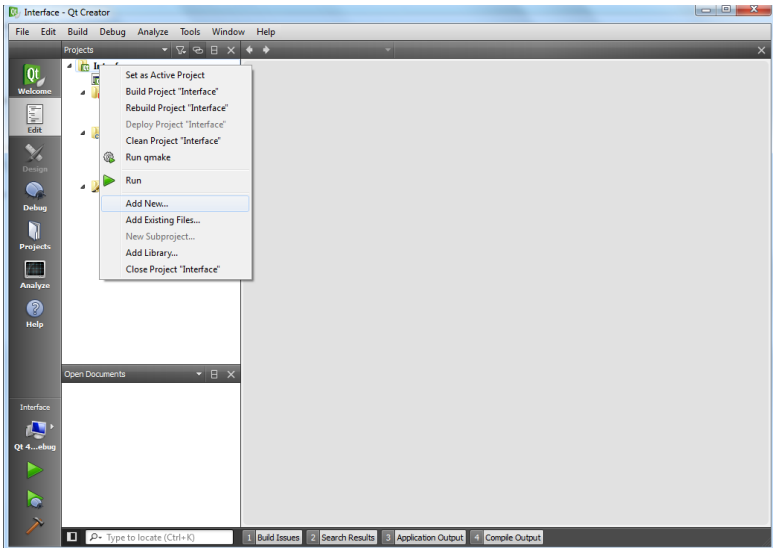

Figura B. 3. Agregando nuevos archivos al proyecto.

4. Seleccionar 'Qt Resource file'.

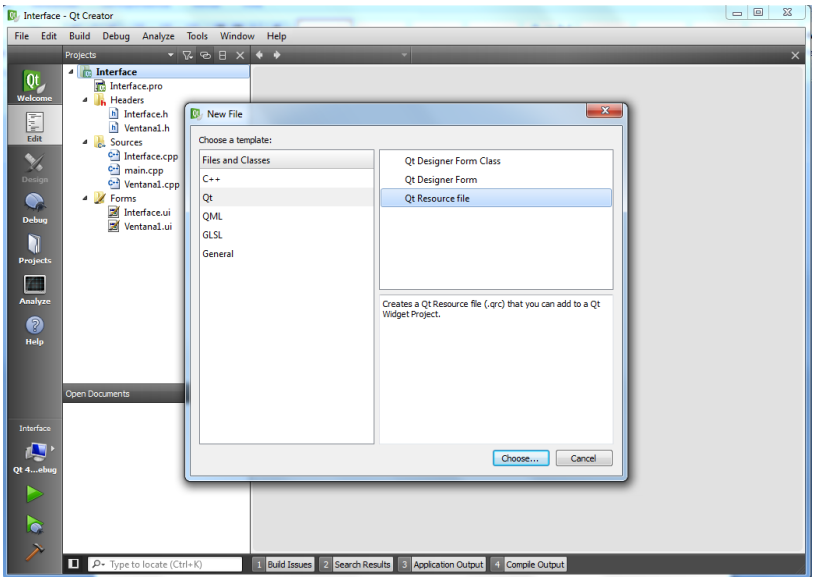

Figura B. 4. Se agrega un archivo de tipo “resource”.

5. Se le asigna un nombre.

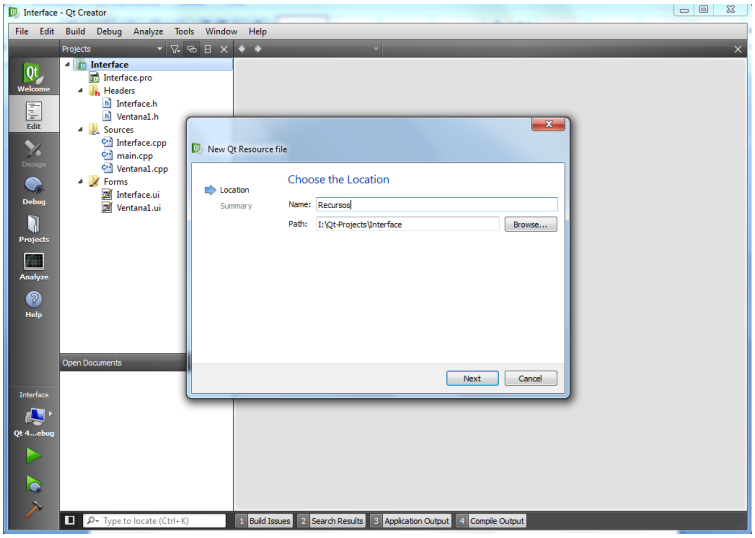

Figura B. 5. Asignando nombre al archivo de recursos.

6. Ahora se da clic en 'add' y luego 'add prefix'.

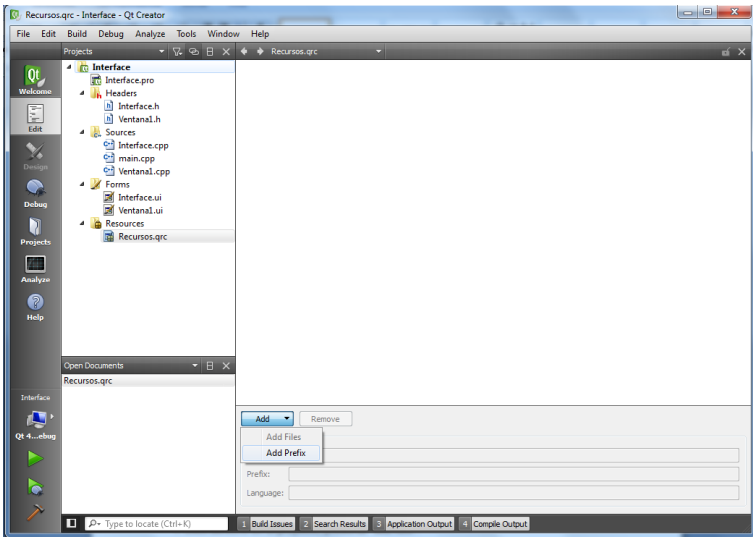

Figura B. 6. Creando directorio de recursos del proyecto.

7. Se asigna el nombre deseado, preferiblemente sencillo y fácil de recordar para cuando haya que asignar recursos desde el código. Luego se presiona en 'add files'.

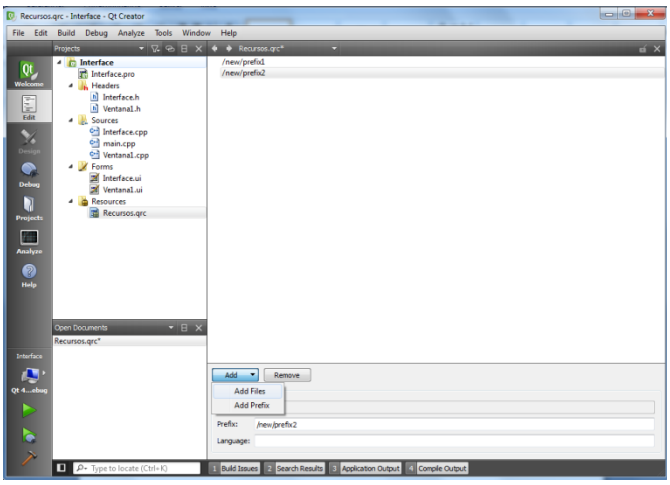

Figura B. 7. Adicionando archivos para usar como recursos.

8. Finalmente se pegan los archivos de Qt en la carpeta del código fuente del proyecto en visual, y se repiten todos los pasos de "Usando CMake".

9. Abrir el archivo .sln generado por CMake dentro de la carpeta “bin”.

10. Se procede a abrir en Qt-Dessigner la interfaz, y se da clic en 'edit resources'.

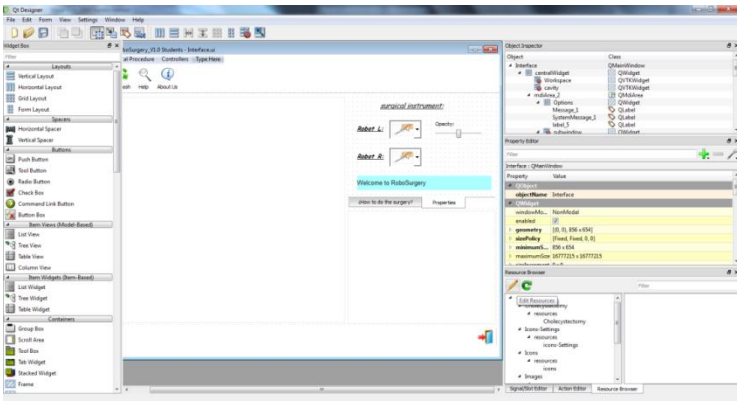

Figura B. 8. Asignando recursos en la interfaz.

11. Se abre el archivo creado en Qt, y automáticamente carga todos los recursos.

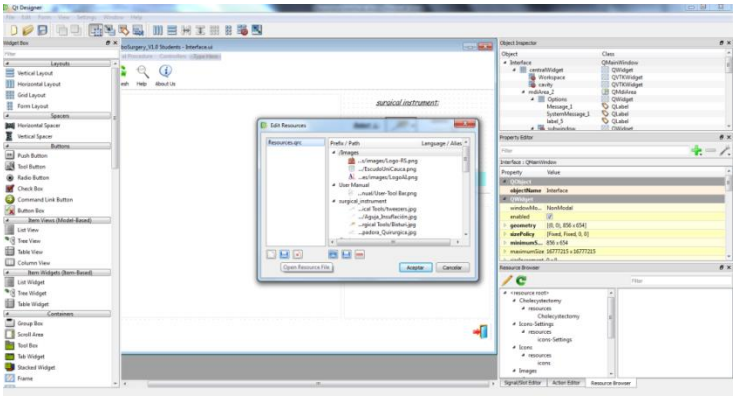

Figura B. 9. Vista de los recursos asignados desde Qt.

El proyecto ya está listo para usar imágenes y crear iconos (recursos).

B.2. Creación de interfaces.

1. Clic derecho en el proyecto, presionar 'add new'.

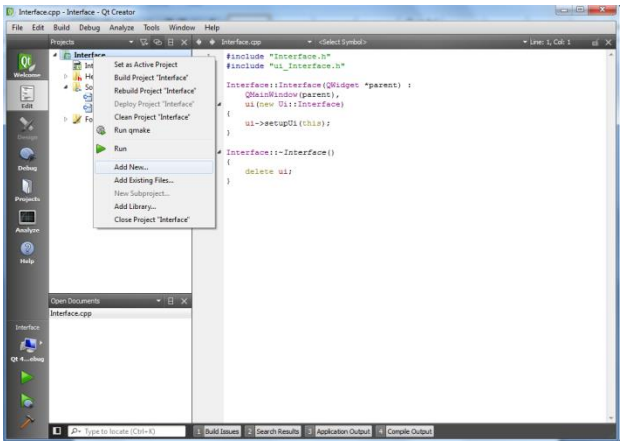

Figura B. 10. Adicionando nuevos archivos.

2. En la siguiente ventana seleccionar 'Qt Designer Form Class'

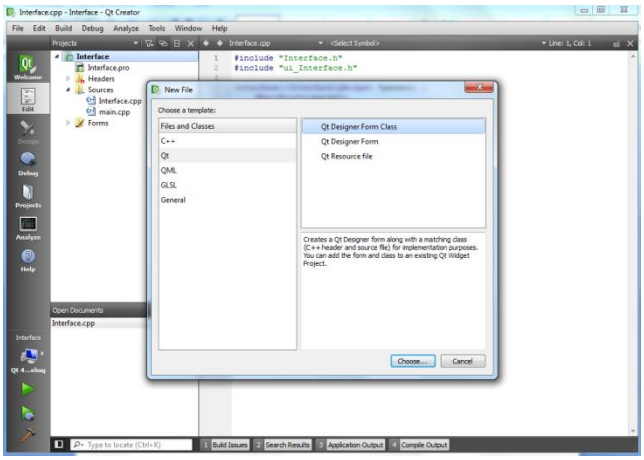

Figura B. 11. Creando archivo tipo “Qt Designer Form Class”.

3. Seleccionar el tipo de widget que se desea crear, por ejemplo 'Dialog without Buttons'.

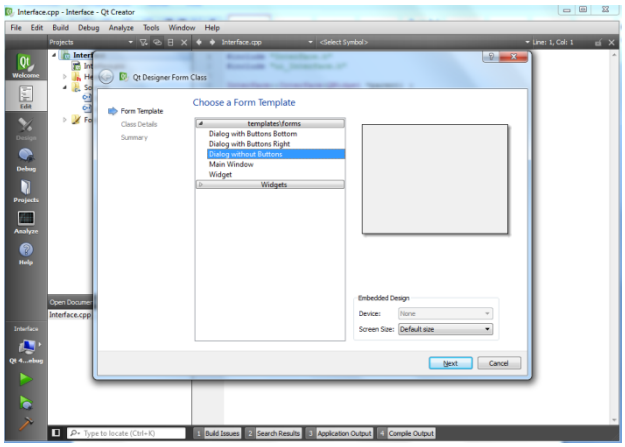

Figura B. 12. Seleccionando tipo de widget para crear.

4. Asignar un nombre a la clase, se recomienda sea tal cual como se llamará dentro del proyecto en visual.

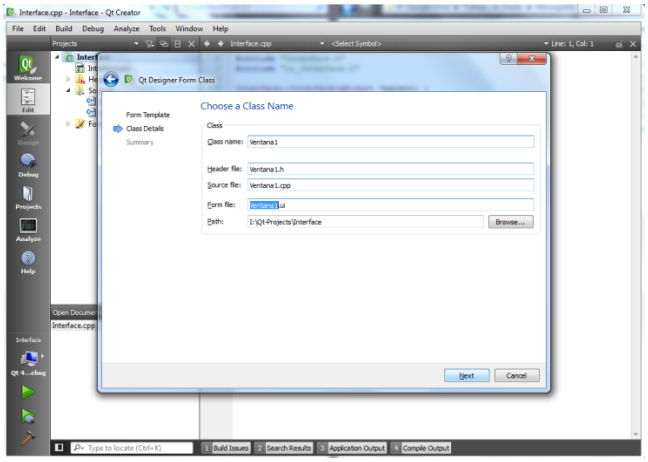

Figura B. 13. Dando nombre al nuevo widget.

5. Se abre la carpeta donde se encuentra el código fuente del proyecto en Qt, se elige los archivos correspondientes a la nueva clase y se copian.

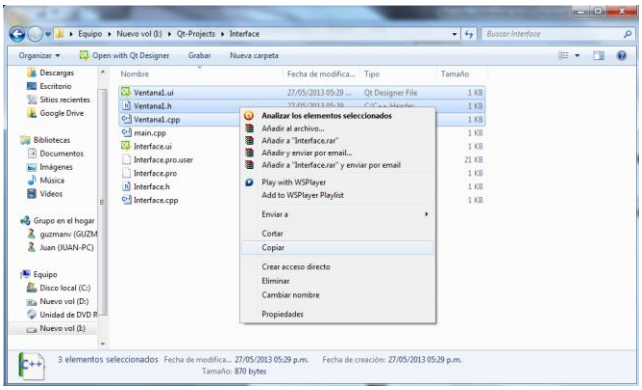

Figura B. 14. Seleccionando los archivos que corresponden a la nueva interfaz.

6. Finalmente se pegan los archivos en la carpeta del código fuente del proyecto en visual, y se repiten todos los pasos de "Usando CMake".

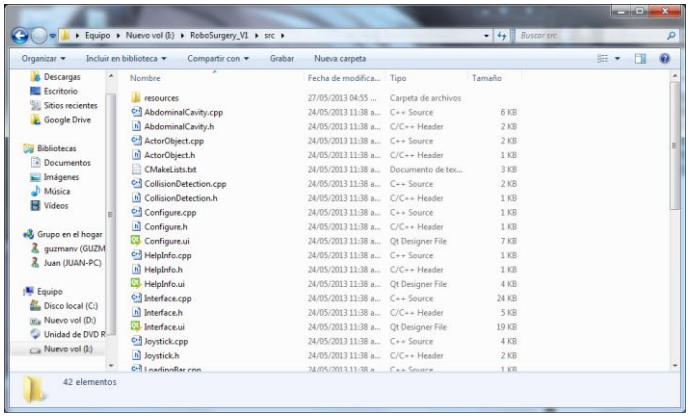

Figura B. 15. Adicionando nuevos archivos al código fuente.

7. Se abre el archivo del proyecto '.sln', en el archivo '.h' de la clase principal, en la sección de archivos de cabecera se incluyen los archivos '.h' de las nuevas interfaces.

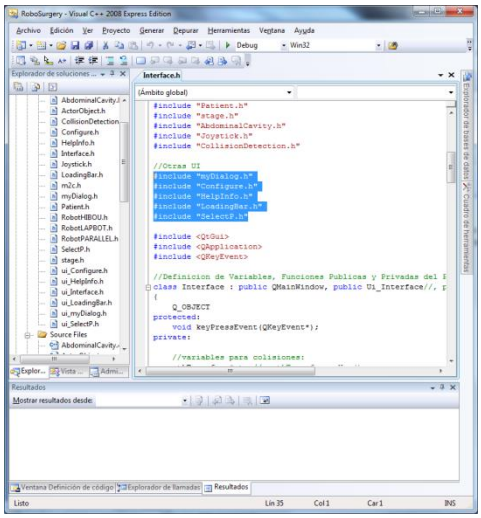

Figura B. 16. Agregando cabeceras de nueva interfaz.

8. Luego entre los atributos privados se crean los objetos respectivos.

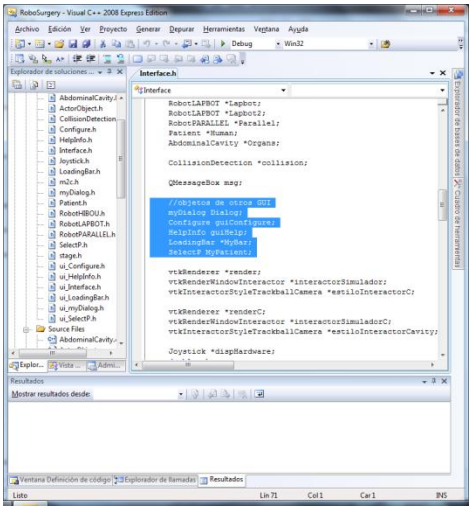

Figura B. 17. Creando objetos correspondientes a las nuevas interfaces.

9. Es importante señalar una diferencias, hay objetos que son puntero, y otros que no. Los que son punteros se utilizan en casos donde se quiere mostrar una interfaz pero la interfaz de atrás debe seguir funcionando.

   Estos se codifican así:

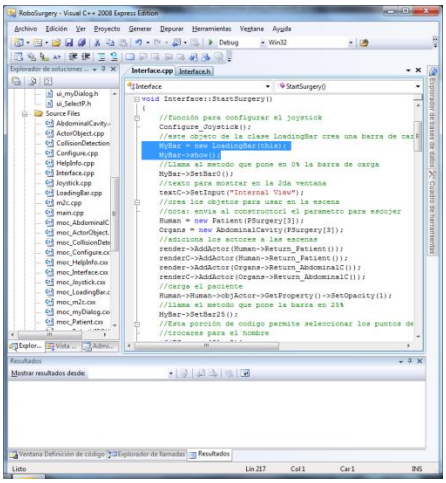

Figura B. 18. Llamando interfaz con puntero.

Los que no llevan puntero, son por ejemplo ventanas de mensaje, que deben 'detener' la ejecución del programa, y dejar solo una activa.

El código va así:

```
void on_actionInfo_triggered(){
Dialog.setModal(true);
Dialog.exec();
}
```

y finalmente se ve asi:

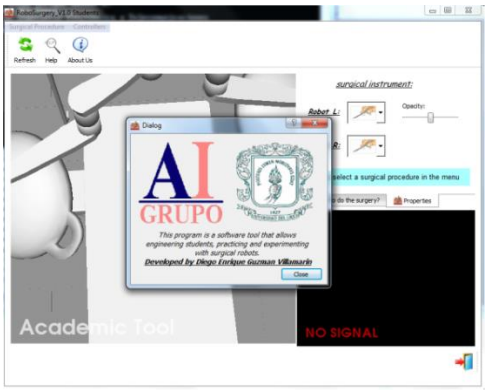

Figura B. 19. Vista de una interfaz adicional.

Anexo C: Creación de librerías compartidas en Matlab [51].

Se crea una biblioteca compartida o DLL para agregar funcionalidad basada en MATLAB a una aplicación que está en desarrollo. Usted selecciona una o más funciones basadas en MATLAB para incluir en la biblioteca compartida, y el compilador de MATLAB genera un archivo binario que se vincula a su aplicación. Las bibliotecas compartidas generadas por el compilador de MATLAB son compatibles con el entorno de desarrollo de Microsoft Visual Studio en Windows y con el C ++ Suite C de GNU / herramienta en la mayoría de las plataformas UNIX.

C.1. La construcción de una biblioteca compartida

Para ilustrar el proceso de creación y uso de bibliotecas compartidas, se usará un algoritmo criptográfico, el **cifrado de Vigenere** . El programe se compone de dos partes:

- libvingenere: Una biblioteca compartida que contiene dos funciones de MATLAB cifrar y descifrar.
- vigenere.cpp: programa principal en C++ que llama a las funciones en libvigenere.

El código fuente para las funciones de MATLAB y el programa principal está disponible en MATLAB central. El paquete de descarga incluye también VigenereDetails.html, que describe la implementación de la cifrado Vigenere en MATLAB.

El comando mcc invoca el compilador MATLAB tanto desde dentro de MATLAB interactivo y en el sistema (DOS o UNIX shell) del sistema. Utilice mcc para construir una biblioteca compartida desde el MATLAB para las funciones **encrypt** y **decrypt** de MATLAB:

```
mcc -v -W cpplib:libvigenere -T link:lib encrypt decrypt
```

Este comando mcc se descompone en cuatro partes:

- `-v` : Activar la salida verbosa.
- `-W Cpplib: libvigenere`: Generar la envolvente del código en C++. Nombre de la biblioteca generada `libvigenere`.

- `-T Link: lib`: Invocar un compilador C / C ++ para crear un archivo binario como biblioteca compartida de la plataforma del código generado.
- `encrypt decrypt` : Coloca las funciones `encrypt` y `decrypt` en la biblioteca compartida. Generar la envolvente C ++ de las funciones para cada uno.

Este comando genera varios archivos. Dos de ellos son relevantes aquí:

`libvigenere.dll`: El binario específico de la plataforma: la biblioteca compartida en sí. En la mayoría de los sistemas Unix, este archivo termina con una extensión `.so`. `libvigenere.so`.

`libvigenere.h`: Declaraciones de la envolvente de las funciones en C++ y de las clases de C ++ tipo de conversión y utilidades.

C.2. La Interfaz Generado

MATLAB Compiler genera muchos diferentes tipos de funciones: inicialización, terminación, de error y de manejo de impresión y, por supuesto, las funciones que se seleccionan para compilar en la biblioteca.

En este ejemplo, el compilador MATLAB genera un punto de entrada C++ para encrypt y `decrypt`. Estas funciones se parecen mucho a las funciones de MATLAB correspondientes.

```
void encrypt(int nargout, mwArray& ciphertext,

             const mwArray& plaintext, const mwArray& key);

void decrypt(int nargout, mwArray& plaintext,

             const mwArray& ciphertext, const mwArray& key);
```

Las funciones generadas difieren de las funciones de MATLAB en dos aspectos significativos:

- Las funciones de C ++ declaran explícitamente los tipos de sus argumentos.
- Las funciones de C ++ regresan void, pasando de nuevo los resultados en los parámetros de salida proporcionados en la lista de argumentos.

Podemos observar la función `encrypt` en detalle. Para la comparación, aquí está la función de MATLAB 's `encrypt` :

```
function ciphertext = encrypt(plaintext, key)
```

La función de MATLAB `encrypt` tiene una salida y dos entradas. La función de C++ `encrypt` tiene cero salidas (el `void` de tipo de retorno) y cuatro entradas. La primera entrada de C++ indica el número de productos solicitados por la persona que llama. Como se ha indicado por la función MATLAB, el número de salidas puede ser cero o uno. Pasando cualquier otro valor se traducirá en un error. La segunda entrada de C++ es el argumento de salida, pasado por referencia. `encrypt` sobrescribirá los datos en este argumento con el mensaje cifrado. El tercer y cuarto argumentos de entrada de C ++ son las entradas de la función, el texto en claro y la clave de cifrado. Ellos se pasan por la constante referencia lo que significa que la función `encrypt` no puede cambiar su contenido.

A diferencia de MATLAB, C++ requiere que todas las variables que se han declarado, tipos inmutables. Las variables en MATLAB tienen tipos, así (matriz, la matriz celular, estructura, etc.), pero el tipo de una variable MATLAB pueden cambiar en cualquier momento. Para adaptarse a este comportamiento dinámico en C++, el compilador de MATLAB proporciona la `mwArray` como un tipo de datos. Todas las funciones generadas por el compilador MATLAB toman como entrada `mwArray` y retorna `mwArray` como salidas. El `mwArray` API le permite crear objetos `mwArray` de la mayoría de los tipos de datos de C++ y extraer datos C++ nativos de un `mwArray` devuelto.

C.3. Llamado de Funciones en una biblioteca compartida

Ahora que he creado una biblioteca compartida, se escribe un programa para llamar a las funciones públicas de la biblioteca. El programa realiza seis tareas:

- Incluye el archivo de cabecera de la biblioteca compartida.
- Analiza los argumentos de línea de comandos.
- Inicializa el Compiler Runtime de MATLAB estado global y el estado de la biblioteca compartida.
- Crea argumentos de entrada; convertir tipos de datos nativo C++ para los tipos de datos de MATLAB.
- Invoca una o más funciones de la biblioteca compartida.
- Cierra la biblioteca y el compilador MATLAB Runtime.

A continuación, se demostrara cómo estas medidas se traducen en el código de ejemplo del programa principal, `vigenere.cpp`.

Paso 1: Todos los programas que utilizan una biblioteca compartida generada por el compilador MATLAB debe incluir el archivo de cabecera de la biblioteca. El archivo de cabecera tiene el mismo nombre base que la biblioteca compilada, `libvigenere` en este caso, y una extensión `.h`.

```
// vingenere.cpp: Encrypt and decrypt using the Vigenere cipher.

#include "libvigenere.h"

#include <iostream>
```

Paso 2: El programa principal analiza la línea de comandos. El primer argumento, un interruptor, determina el tipo de acción: `-e` significa cifrar, `-d` descifrar. El segundo argumento es el mensaje, y el tercero, la clave.

```
int main(int ac, const char *av[])

{

    // Encrypt or decrypt? Determined by command line switch

    bool enc = strcmp(av[1], "-e") == 0;
```

Paso 3: Iniciar el tiempo de ejecución y comenzar la biblioteca antes de llamar a cualquiera de las funciones exportadas de la ejecución o la biblioteca. Si existe un error al inicializar hará que su programa se bloquee. Compruebe para el éxito de la compilación (los inicializadores regresan `false` si fallan) los mensajes de error según sea necesario.

```
    // Initialize the MATLAB Compiler Runtime global state

    if (!mclInitializeApplication(NULL,0))

    {

        std::cerr << "Could not initialize the application properly."

                  << std::endl;
```

```
        return -1;

    }

    // Initialize the Vigenere library

    if( !libvigenereInitialize() )

    {

        std::cerr << "Could not initialize the library properly."

                  << std::endl;

        return -1;

    }
```

Paso 4: Convertir los `strings` C ++ desde la línea de comandos (el mensaje y la clave) en cadenas de MATLAB creando objetos `mwArray`. Estas declaraciones no pueden aparecer antes de la inicialización hecha en el paso 3.

```
// Must declare all MATLAB data types after initializing the

// application and the library, or their constructors will fail.

    mwArray text(av[2]);

    mwArray key(av[3]);

    mwArray result;
```

Paso 5: Invocar las funciones exportadas. `encrypt` o `decrypt` como se indica por el interruptor de línea de comandos. Tenga en cuenta que las funciones de C ++ tienen el mismo nombre que sus contrapartes de MATLAB, y que todos los valores de retorno se deben pasar en modo de referencia. Los `mwArray` define la clase.

```
    // Initialization succeeded. Encrypt or decrypt.

    if (enc == true)

    {
```

```
        // Encrypt the plaintext text with the key.
        // Request one output, pass in two inputs
        encrypt(1, result, text, key);
    }
    else
    {
        // Decrypt the ciphertext text with the key.
        // Request one output, pass in two inputs
        decrypt(1, result, text, key);
    }
    std::cout << result << std::endl;
```

Paso 6: Apague la biblioteca y el tiempo de ejecución, en el orden inverso de la inicialización (biblioteca, luego tiempo de ejecución).

```
// Shut down the library and the application global state.
    libvigenereTerminate();
    mclTerminateApplication();
```

Este proceso permite el uso también de funciones tipo Mex, que contienen un grupo funciones protegidas pero que permiten su uso externo.

Anexo D: Manual de Usuario Elax1.

Esta guía indica al usuario como hacer uso adecuado de la herramienta **Elax1**, además de explicar el funcionamiento de los elementos dispuestos en la interfaz.

1. Iniciar la aplicación.

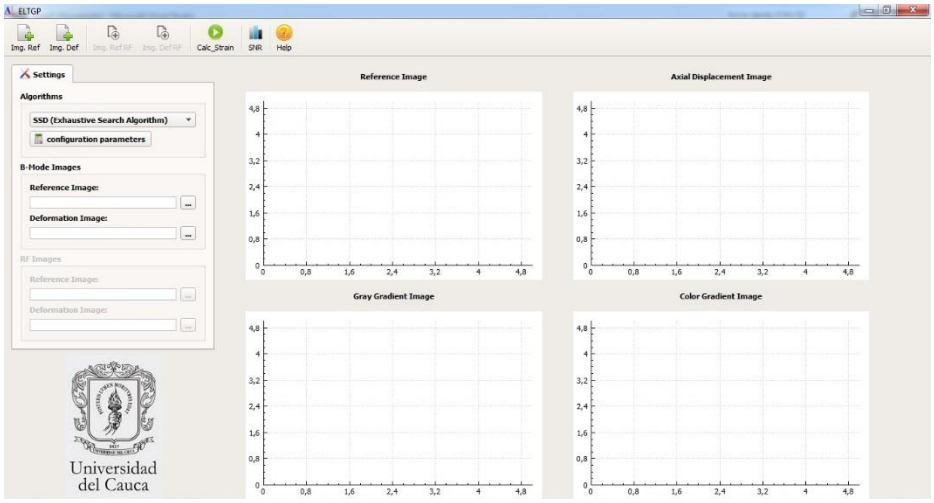

Figura D. 1. Interfaz de la aplicación Elax1

2. Seleccionar el algoritmo a utilizar para el cálculo de la deformación usando la pestaña superior izquierda

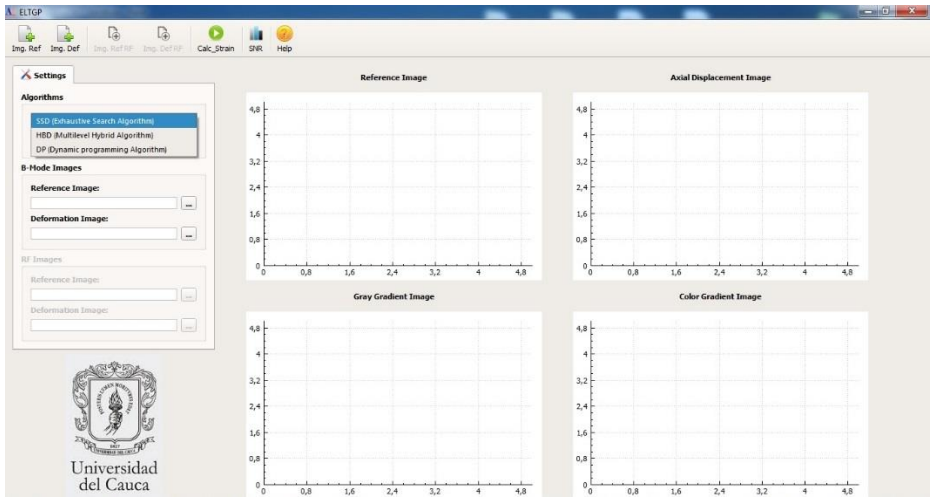

Figura D. 2. Selección algoritmo en la aplicación Elax1

Se puede seleccionar entre tres algoritmos: SSD, HBD y AM2D. Para su correcto funcionamiento el algoritmo SSD requiere de las imágenes en modo B de los tejidos a analizar. El algoritmo HBD requiere de las imágenes en modo B y las imágenes RF puras y el algoritmo AM2D requiere solo las imágenes RF puras.

3. Una vez se ha seleccionado el algoritmo, se procede a cargar los archivos que contienen los datos de las imágenes de referencia y deformación. Esto se puede realizar de dos formas. La primera es mediante el botón con puntos suspensivos ubicado en panel de selección de imágenes como en la figura D.3.

Figura D. 3. Cargar archivos mediante panel de selección

La segunda forma es mediante los botones ubicados en la parte superior en la barra de herramientas.

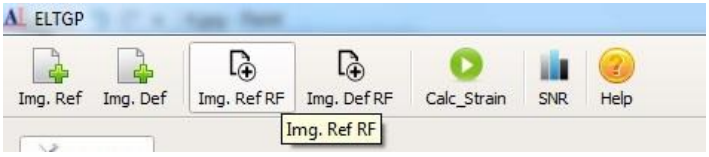

Figura D. 4. Cargar archivos median la barra de herramientas

De acuerdo con el algoritmo seleccionado se habilitan las opciones de carga de imágenes, si es SSD habilita las imágenes en modo B, si es AM2D habilita las imágenes RF y si es HBD habilita las dos opciones. Los archivos de lectura tienen formato YML.

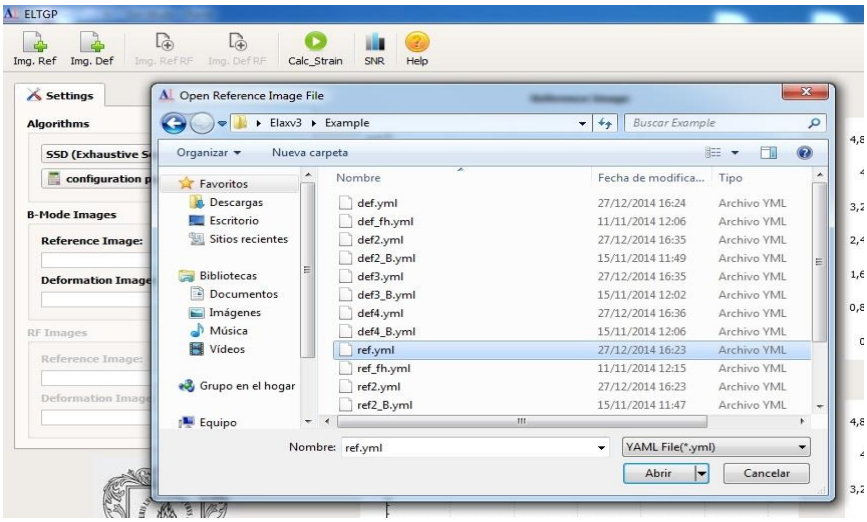

Figura D. 5. Archivos usados en la experiencia en la aplicación Elax1

En caso de cargar erróneamente los archivos el programa le advertirá y solicitara nuevamente cargar los archivos.

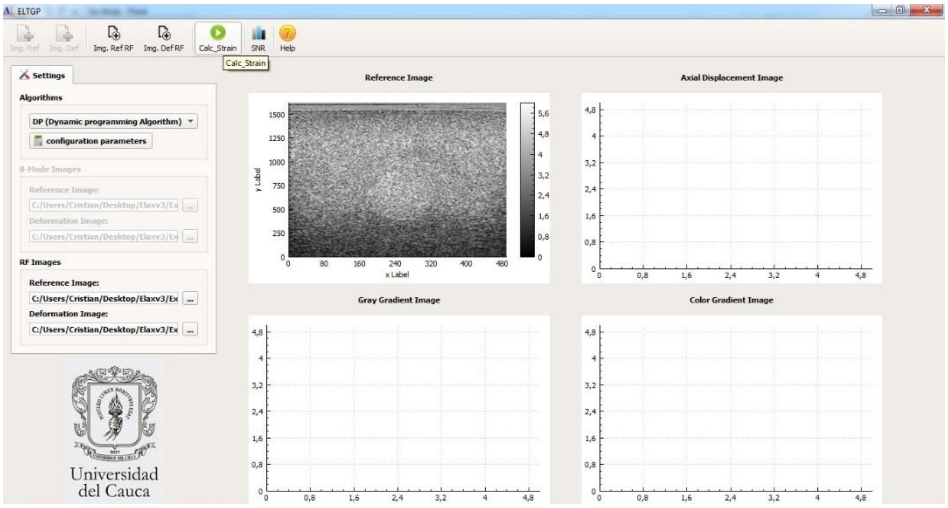

Figura D. 6. Vista previa de archivos cargados en la aplicación

4. Una vez se hayan cargado los archivos de interés, se procede a calcular la imagen de deformación mediante el botón **Calc_Strain.**

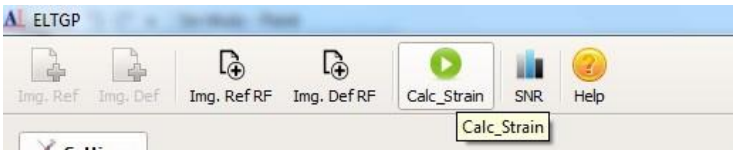

Figura D. 7. Calculo de la imagen de deformación

Aquí, la aplicación se encarga de hacer los cálculos correspondientes para mostrar los resultados del elastograma.

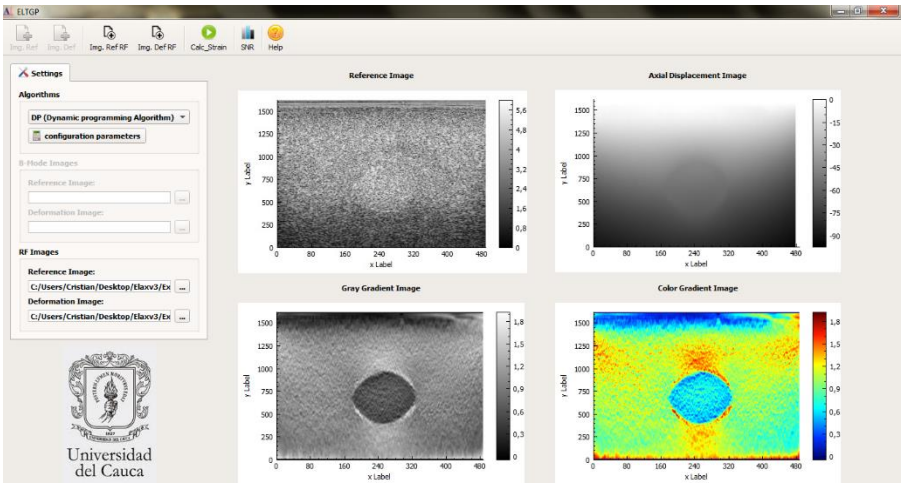

Figura D. 8. Vista previa resultados obtenido por el cálculo de deformación

La ventana muestra cuatro imágenes que corresponden a: la imagen de referencia en modo B (parte superior izquierda), la imagen de desplazamiento axial (parte superior derecha), la imagen de deformación axial en escala de grises (parte inferior izquierda), y la imagen de deformación axial en escala de colores (parte inferior derecha).

Para los ejemplos de base con los que se trabajan, los parámetros utilizados por los algoritmos están con valores por defecto para su correcto funcionamiento. Si desea hacer pruebas puede realizar cambios a dichos parámetros mediante el botón **configuration parameters**.

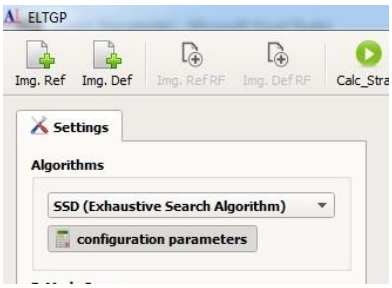

Figura D. 9. Configuración de parámetros de los algoritmos

Esto habilitará una ventana donde se podrán hacer las modificaciones.

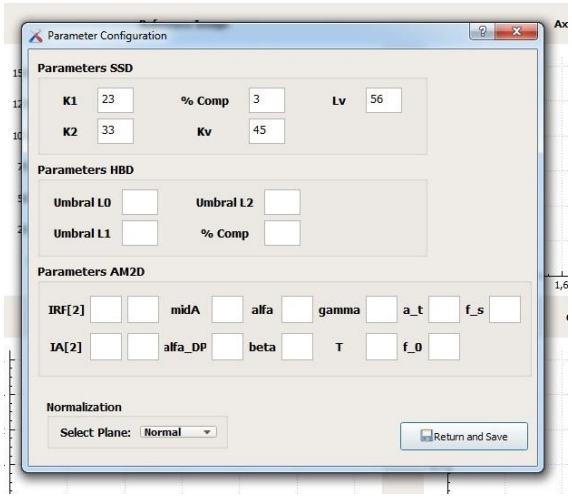

Figura D. 10. Ventana de configuración de parámetros

Adicional al cambio de parámetros se podrá seleccionar el plano de normalización a usar. Si selecciona el plano normal no aplicara cambios a los resultados.

Figura D. 11. Selección del plano de normalización en la ventana de parámetros

5. Para poder tener un análisis cuantitativo de los resultados obtenidos, se hace uso del cálculo de señal a ruido, para ello dar clic sobre el botón **SNR** de la barra de herramientas.

Figura D. 12. Calculo de SNR en la barra de herramientas

Este desplegará una ventana donde se hace un cálculo de señal a ruido sobre la imagen de deformación obtenida y muestra un histograma que contiene la información del conteo de concentración de pixeles sobre la imagen como se observa en la figura D.13.

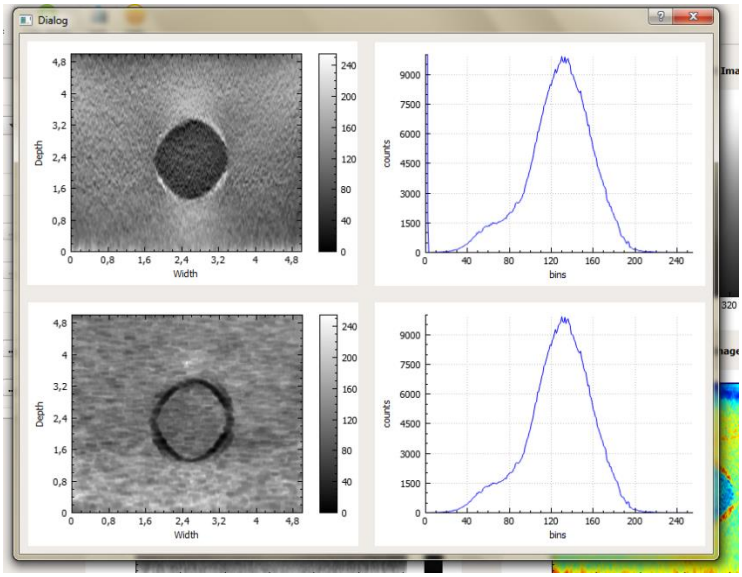

Figura D. 13. Ventana de resultados calculo SNR

Esta opción se puede utilizar con los resultados obtenidos en cada algoritmo que usa la aplicación.

6. Por último se tiene la opción **Help** en la barra de herramientas, la cual despliega una ventana con información sobre los desarrolladores y un link que abre este manual de usuario.

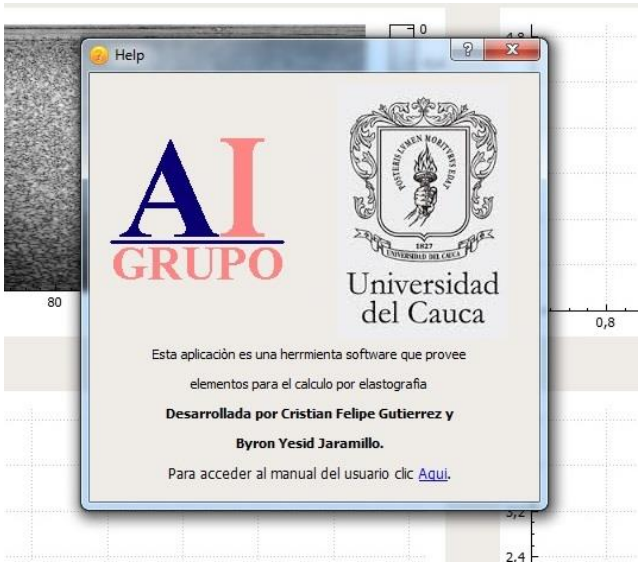

Figura D. 14. Ventana Help de la aplicación Elax1