9. Anexos

9.1 Nociones básicas sobre Visión por Computador

Modelo Pinhole

El modelo básico de una cámara Pinhole consta de un centro óptico C, en donde convergen todos los rayos de la proyección, y un plano de imagen R en el cual la imagen es proyectada. El plano de imagen está ubicado a una distancia focal f del centro óptico y perpendicular al eje Z.

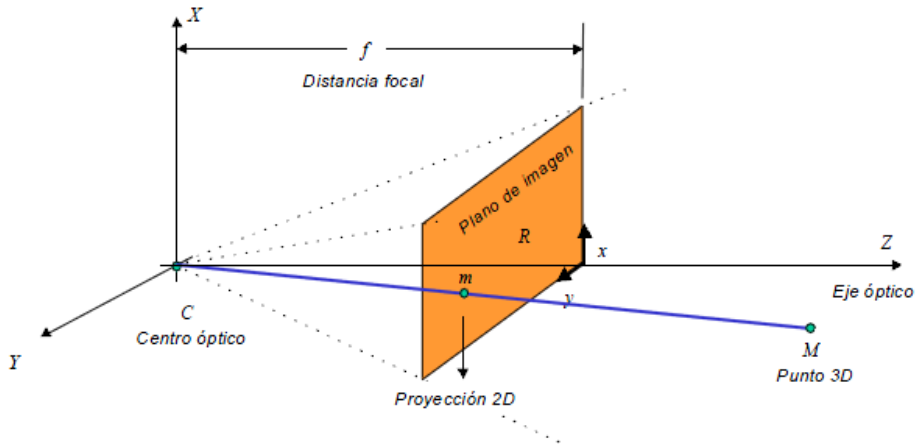


Figura 1: Modelo geométrico de una cámara Pinhole.

Un punto 3D M es proyectado en el plano de imagen como m. El punto 2D m se define como la intersección de la recta (C, M) con el plano R.

Suponiendo que las coordenadas de los puntos M y m son $(X, Y, Z)^T$ y $(x, y)^T$ respectivamente y f es la distancia focal de la cámara, se puede encontrar la siguiente relación:

$$Z\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} fX \\ fY \\ Z \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & f & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

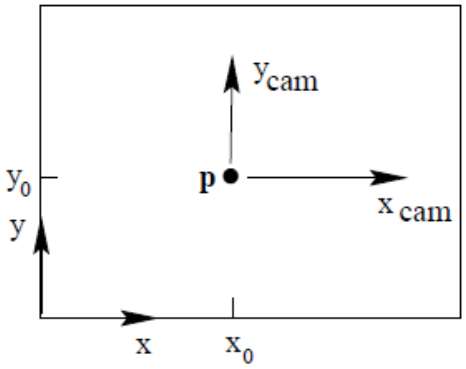
o también:

$$\lambda m = PM$$

Donde $M = [X\ Y\ Z\ 1]^T$ y $m = [x\ y\ 1]^T$ son las coordenadas homogéneas de M y m, y P la matriz 3x4 denominada matriz de proyección de perspectiva de la cámara. El factor λ es un factor de escala para mantener la igualdad y es igual a Z.

<u>Parámetros intrínsecos de una cámara</u>

$$k_x\ x_{\text{cam}} = x - x_0$$
$$k_y\ y_{\text{cam}} = y - y_0$$



x e y equivalen a las coordenadas básicas de un punto en pixeles, mientras que x_{cam} e y_{cam} corresponden a esos mismos puntos referenciados con respecto al centro de la cámara en mm. k_x y k_y están expresados en [pixeles/mm] y expresan a cuantos pixeles equivale un mm en el eje x y el eje y.

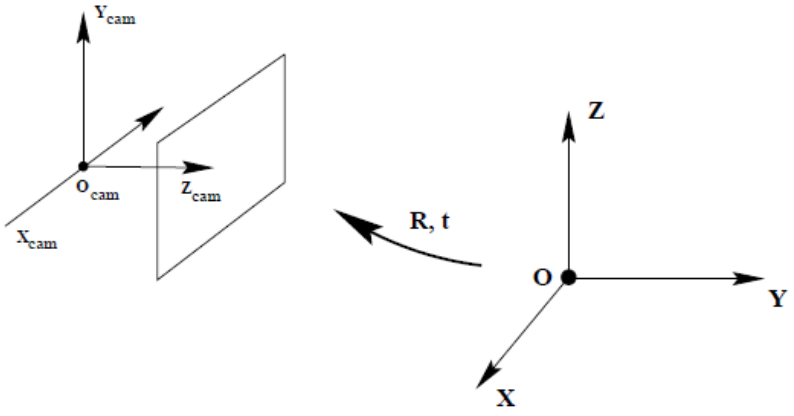
$$\mathbf{x} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \frac{1}{f} \begin{bmatrix} \alpha_x & & x_0 \\ & \alpha_y & y_0 \\ & & 1 \end{bmatrix} \begin{pmatrix} x_{\text{cam}} \\ y_{\text{cam}} \\ f \end{pmatrix} = \mathrm{K} \begin{pmatrix} x_{\text{cam}} \\ y_{\text{cam}} \\ f \end{pmatrix}$$

α_x equivale a fk_x y α_y equivale a fk_y. K corresponde a la matriz de parámetros intrínsecos de la cámara y es la que nos permite transformaciones entre un punto de la imagen y un rayo del espacio euclideo.

En la matriz K aparecen cuatro parámetros:

- La escala de la imagen en la dirección x e y, que corresponde a α_x y α_y.
- El punto principal (x_0, y_0), que corresponde al punto en donde el centro óptico se cruza con el plano imagen.

<u>Parámetros extrínsecos de una cámara</u>



Los parámetros extrínsecos de una cámara corresponden a la matriz de rotación R y vector de translación t del eje de coordenadas de la cámara O_{cam} con respecto a una referencia mundo O.

$$\begin{pmatrix} X_{cam} \\ Y_{cam} \\ Z_{cam} \\ 1 \end{pmatrix} = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

Para pasar las coordenadas de un punto referenciado con respecto a la referencia mundo a coordenadas de la cámara basta con aplicar la ecuación previa.

Concatenando las tres matrices vistas hasta ahora obtenemos:

$$x = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = K \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = K\ [R|t]X$$

las cuales definen una matriz de proyección 3x4 desde el espacio euclideo a la imagen como:

$$x = PX \qquad P = K[R|T] = KR[I|R^T t]$$

Cámara proyectiva

Una cámara desde el punto de vista proyectivo es una relación lineal entre puntos con coordenadas homogéneas.

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \begin{bmatrix} \quad \mathrm{P}\ (3 \times 4) \quad \end{bmatrix} \begin{pmatrix} \mathrm{X} \\ \mathrm{Y} \\ \mathrm{Z} \\ 1 \end{pmatrix}$$

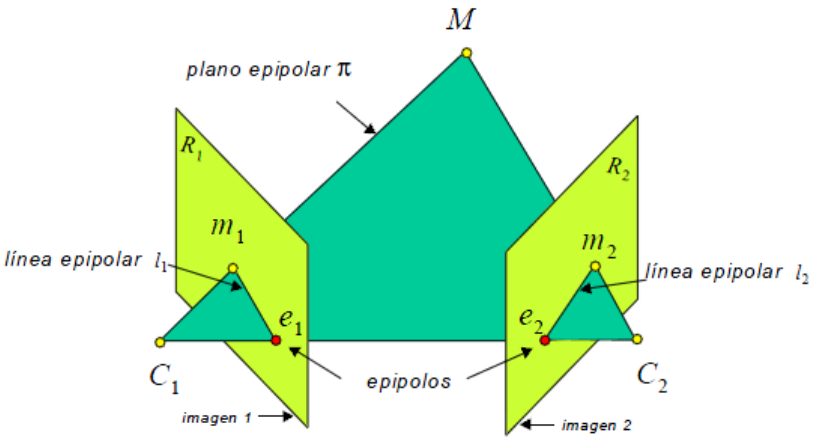
Image Point Scene Point

$$\mathbf{x} = \mathrm{P}\ \mathbf{X}$$

P tiene 11 grados de libertad, el centro de la cámara equivale a la última columna de P.

Geometría epipolar

Dadas dos imágenes distintas de una escena con centros ópticos C_1 y C_2 en donde m_1 y m_2 corresponden al mismo punto cuya proyección 3D es M, se denomina línea epipolar a la proyección de la línea que une C_1 con M sobre el plano R_2. La característica de la línea epipolar l_2 obtenida a partir de m_1 es que sobre dicha línea epipolar debe situarse m_2.



La geometría epipolar depende únicamente de la posición y orientación de las cámaras y de sus parámetros intrínsecos. Todas las líneas epipolares se cruzan en los epipolos.

Matriz Fundamental

La matriz fundamental cumple las siguientes propiedades:

1) Las representaciones homogéneas de las líneas epipolares l_1 y l_2 se definen como:

$$l_2 = Fm_1$$
$$l_1 = Fm_2$$

2) La restricción epipolar es:

$$m_2^T F m_1 = 0$$

3) La matriz F es homogénea, ya que kF para $k \neq 0$ también puede ser utilizada en los cálculos anteriores.
4) El determinante de F es cero.
5) F solo tiene siete grados de libertad, por lo que solo siete (de los nueve) elementos de F son linealmente independientes, los otros dos elementos pueden ser calculados como función de los otros siete.
6) La matriz F es constante, no depende de m_1, m_2 y M, solo depende de las matrices de proyección P y P'

9.2 Instalación y ejecución de Bundler

Dirigirse a la dirección http://phototour.cs.washington.edu/bundler/ y descargarse la última versión de bundler. En nuestro caso para este proyecto se ha utilizado la versión 0.3 que era la última hasta la fecha.

Existen dos versiones, la binary (ya están generados los ejecutables) y la source (solo está el código y hay que compilarla). Si se quiere utilizar la versión source habría que compilar previamente el código con por ejemplo Visual Studio abriendo el proyecto desde el archivo Bundler.sln de la carpeta vc++, también sería posible compilarlo desde cygwin utilizando la función make. Para entornos Linux habría que instalar las dependencias como son imageMagic antes de intentar compilar el código mediante la función make. A continuación se detalla la instalación de la versión binary en un entorno Windows para su ejecución bajo Cygwin.

Binary Versión:

- Descomprimir el archivo bundler-v0.3-binary.zip en un directorio.
- Copiar el archivo siftWin32.exe a la carpeta .../bundler-v0.3-binary/bin, el cual se puede descargar desde la siguiente dirección http://www.cs.ubc.ca/~lowe/keypoints/siftDemoV4.zip.
- Instalar Cygwin con al menos las librerías gcc, make, imageMagic.
- Copiar las imágenes con las que se quiere ejecutar bundler (por ejemplo las que hay en la carpeta .../bundler-v0.3-binary/examples/kermit) al directorio bundler-v0.3-binary.
- Ejecutar Cygwin y dirigirse a la carpeta .../bundler-v0.3-binary
- Escribir en el programa "*sh ./RunBundle.sh*".
- Si todo ha ido bien en la carpeta ... /bundler-v0.3-binary/bundle deberan aparecer los archivos de salida.
- Para hacer más pruebas con otras imágenes ejecutar antes desde Cygwin en el directorio bundler-v0.3-binary la instrucción "*make clean*" para borrar los ficheros que se han creado en la anterior ejecución.

9.3 Instalación y ejecución de PMVS

Dirigirse a la dirección http://grail.cs.washington.edu/software/pmvs/ y descargarse la última versión de PMVS, en nuestro caso se descargo la siguiente versión pmvs-2.tar, aunque recientemente ha salido una versión corregida de esta. Una vez descargado y descomprimido se procede a realizar los siguientes pasos:

- Instalar Cygwin con al menos las librerías Libboost, jpeg, gsl-devel, libjpeg-devel, libXext-devel.
- Dirigirse a la ruta C:\cygwin\usr\include\boost-1_33_1 (o la correspondiente a la instalación de cygwin) y mover la carpeta boost a la dirección C:\cygwin\usr\include\ quedando así la estructura de la siguiente forma C:\cygwin\usr\include\boost. Esta modificación es necesaria ya que el instalador de cygwin copia en una ruta errónea la librería boost impidiendo que luego cygwin sea capaz de localizarla.
- Descargarse los ficheros f2c.h, clapack.h, blaswrap.h de la librería clapack de la página http://www.netlib.org y meterlos en una carpeta llamada clapack.
- Mover la carpeta clapack a la dirección C:\cygwin\usr\include\. Esta librería no se puede instalar con el instalador de Cygwin por lo que esta es la única forma de incluirla en cygwin.
- Ejecutar cygwin y escribir en el directorio .../pmvs/program/main la instrucción "*make clean*" para borrar posibles archivos de compilación generados y quedarnos solo con los fuente.
- Escribir make depend (Esta instrucción compila las librerías dependientes y necesarias para que funcione el pmvs-2).
- Escribir make (construye el ejecutable pmvs2.exe).
- Ejecutar la instrucción ./pmvs2 ../../data/hall/ option.txt para probar si funciona. Si no da errores de dependencias y se empieza a ejecutar cancelar con Ctrl+C.
- Para ejecutar otros modelos con pmvs2 se deberá utilizar el ejecutable Bundle2PMVS que viene en el bundler y después ejecutar el script sh que nos

indica. Esto generara una carpeta llamada pmvs en donde se guardan los archivos que hay que proporcionarle a pmvs2. Además de estos datos se debe escribir un archivo option.txt (con los parámetros básicos de ejecución) el cual se puede copiar del ejemplo hall y modificar para indicar el numero de imágenes que tiene que tratar.

9.4 Instalación y ejecución de Qt en Visual Studio

A continuación se procede a describir cada uno de los pasos realizados para conseguir instalar y ejecutar Qt con éxito en Visual Studio 2008.

Dirigirse a la dirección http://qt.nokia.com/downloads y seleccionar la versión LGPL/Free, una vez ahí descargarnos la versión SDK para Windows. En nuestro caso se utilizo la siguiente versión *qt-sdk-win-opensource-2009.04.exe*. Descargar también el plugin Visual Studio Add-in de la dirección http://qt.nokia.com/downloads/visual-studio-add-in.

Ya descargados proceder a instalar ambos ejecutables. Una vez instalados nos dirigimos a la ventana de comandos de Visual Studio situada generalmente en Inicio -> Todos los programas -> Microsoft Visual Studio 2008 -> Visual Studio Tools -> Símbolo del sistema de Visual Studio 2008. Desde la ventana de comandos vamos a la ruta C:\QT\<versión>\qt y escribimos la siguiente instrucción "*configure -platform win32-msvc2008*". Tras la ejecución escribir "*nmake*", la ejecución de estas 2 instrucciones puede llevar entorno a 3 o 4 horas.

Añadir la siguiente línea a la variable de entorno PATH del sistema operativo "*C:\Qt\<versión>\qt\bin*", en nuestro caso "*C:\Qt\2009.04\qt\bin*".

Abrir Visual Studio y en el menu Qt -> Qt Options seleccionar Add y añadir la ruta en la que se ha instalado Qt, en nuestro caso "*C:\Qt\2009.04\qt\bin*".

Tras realizar todos estos pasos ya podemos crear, compilar y ejecutar proyectos en Qt. Para crear un nuevo proyecto en Qt basta con seleccionar en Visual Studio Archivo -> Nuevo -> Proyecto la opción Qt Application. Si se desea añadir al proyecto algunos módulos de Qt como son OpenGL, se deberá seleccionar el menú Qt -> Qt Project Settings y en la pestaña Qt Modules marcar OpenGL library.

Hay que tener en cuenta que al ejecutar la aplicación que estamos programando en Qt el PATH que hemos establecido anteriormente y el plugin Visual

Add-in indica a Visual Studio donde encontrar los archivos .dll, pero si queremos ejecutarlo en otro ordenador habrá que copiar los archivos .dll correspondientes junto al ejecutable.

9.5 Instalación de OpenCV para Visual Studio

Dirigirse a la dirección http://opencv.willowgarage.com/wiki/ y descargar la última versión de OpenCV, en nuestro caso *OpenCV-2.0.0a-win32.exe*.

Una vez descargado proceder a la instalación. Tras instalarlo satisfactoriamente nos descargamos también la aplicación cmake de la siguiente dirección http://www.cmake.org/files/v2.8/cmake-2.8.1-win32-x86.exe. Instalamos cmake y ejecutamos CMake GUI. Ya en CMake GUI, en *"Where is the source code"*, seleccionamos *C:\OpenCV2.0* y posteriormente creamos un directorio llamado vs2008 en *C:\OpenCV2.0*. Volvemos a CMake GUI, y en *"Where to build the binaries"*, seleccionamos el directorio *C:\OpenCV2.0\vs2008*. Tras realizar estos pasos pinchamos en configurar y seleccionamos *"Visual Studio 2008"*. Una vez terminado seleccionamos las opciones que queremos y le volvemos a dar a configurar. Para finalizar seleccionamos generar.

Abrimos el proyecto *"C:\OpenCV2.0\vs2008\OpenCV.sln"* en Visual Studio y lo compilamos en modo Debug y Release.

Añadir las siguientes líneas al PATH del sistema *C:\OpenCV2.0\vs2008\bin\Debug*; *C:\OpenCV2.0\vs2008\bin\Release*;

Ejecutar Visual Studio y seleccionar el menú Herramientas -> Opciones, dirigirse a la pestaña proyectos y soluciones -> Directorios de VC++ y añadir en archivos ejecutables C*:\OpenCV2.0\vs2008\bin\Debug* y *C:\OpenCV2.0\vs2008\bin\Release*.

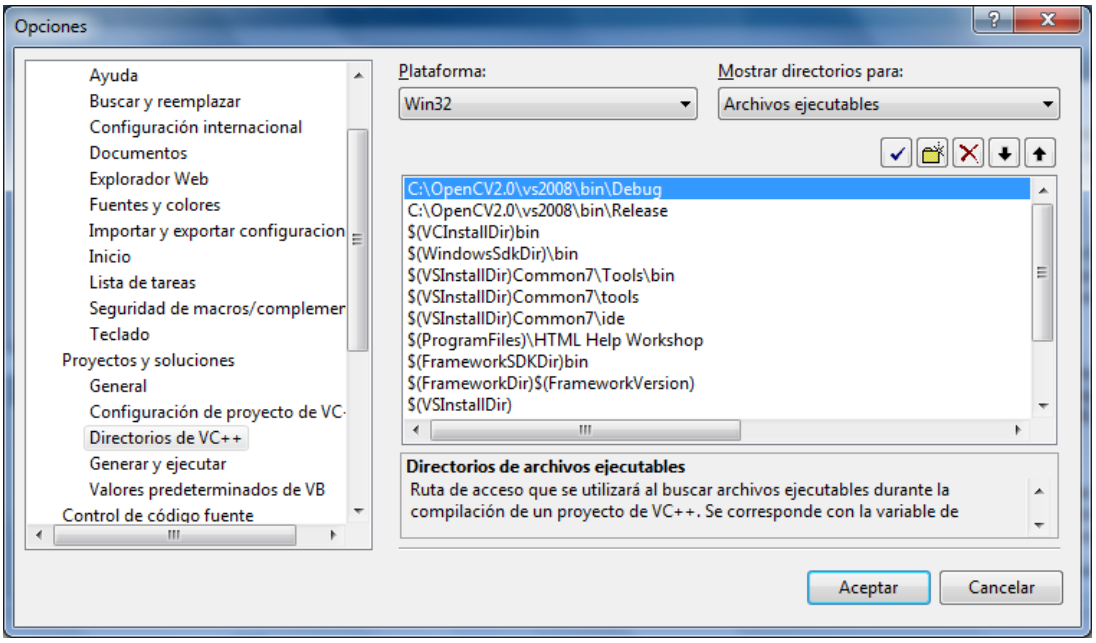


Figura 2: Añadir las rutas ejecutables de OpenCV a VS.

En el mismo lugar seleccionar archivos de inclusión y añadir *C:\OpenCV2.0\include\opencv*. Por último en archivos de biblioteca añadir *C:\OpenCV2.0\vs2008\lib\Debug* y *C:\OpenCV2.0\vs2008\lib\Release*.

Cuando queramos utilizar en un proyecto OpenCV es necesario especificarle al proyecto donde se localizan los archivos .lib necesarios para su compilación. Para ello nos dirigimos al menú proyecto -> propiedades de <nombre del proyecto> y en la pestaña Propiedades de configuración -> Vinculador -> Entrada añadimos en dependencias adicionales highgui200.lib cvaux200.lib cv200.lib cxcore200.lib.

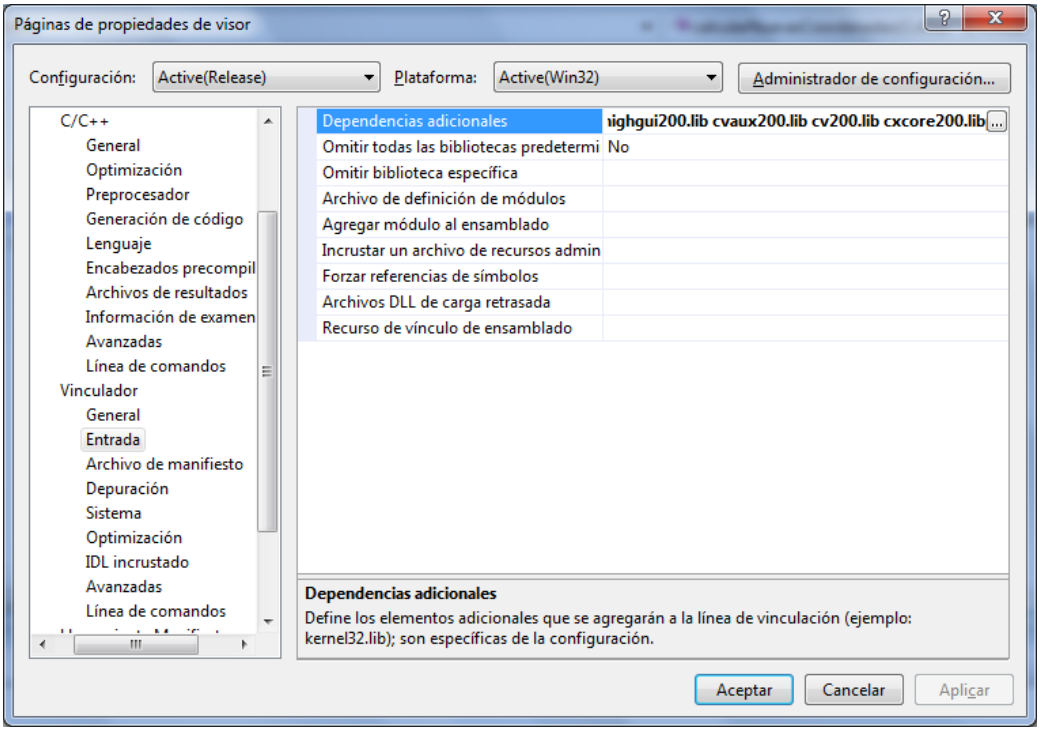


Figura 3: Ejemplo de .lib añadidos a un proyecto que utiliza OpenCV.

9.6 Manual de usuario de la aplicación

9.6.1 Estructura básica de las carpetas de los modelos y objetos

A lo largo de este proyecto se decidió crear una carpeta o base de datos en la cual se encontrarían a su vez las diferentes carpetas de las escenas y objetos. Cada carpeta de la escena y objeto comparten una estructura en común la cual es necesaria para el correcto funcionamiento de la aplicación. A continuación se muestran los diferentes archivos necesarios.

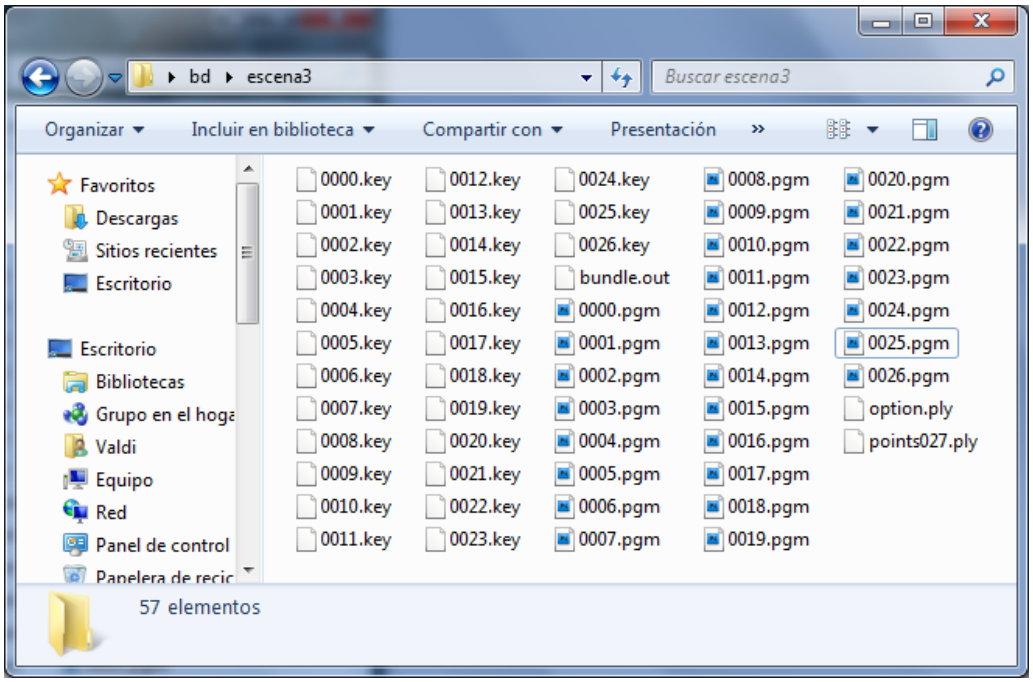


Figura 4: Ejemplo de la estructura de las escenas y objetos.

- Los archivos .pgm corresponden a los archivos .jpg empleados para obtener el modelo en PMVS y a los cuales ya se les ha quitado la distorsión con la función RadialUndistort de Bundler. En cuanto a la conversión de los archivos .jpg a formato .pgm se ha decidido por comodidad programar un script para ejecutar desde Cygwin que convierte todos los archivos .jpg de una carpeta a .pgm.

- Los archivos .key de cada imagen han sido obtenidos de los archivos *.key.gz proporcionados por Bundler. Hay que tener en cuenta que los archivos .pgm tienen como nombre la numeración 0000 hasta el numero de imágenes debido a que al pasar la función Bundle2PMVS de Bundler renombra las imágenes, mientras que los archivos .key extraídos de Bundler tienen el nombre de las imágenes previas y por lo tanto hay que renombrarlos al mismo nombre que tienen ahora las imágenes. Otra cosa a tener en cuenta es que Bundler puede no haber sido capaz de posicionar una cámara y por lo tanto esa imagen no ha sido añadida a la base de datos por lo que el fichero .key de esa imagen tampoco hay que añadirlo ni renombrarlo a la base de datos.

- Además de las imágenes y los puntos de interés SIFT es necesario tener también el archivo bundle.out obtenido de la ejecución de Bundler ya que en este se almacenan las posiciones de las cámaras.

- Por último obviamente se ha de tener el archivo .ply calculado con PMVS que corresponde al modelo de la escena u objeto.

9.6.2 Como desplazarse en el modelo cargado

Una vez hayamos cargado una escena en el visor podemos desplazarnos por esta de la siguiente manera:

- Rotación en el eje X y eje Y mediante el botón izquierdo del ratón.
- Rotación en el eje Z mediante la barra de desplazamiento inferior al programa.
- Desplazamiento en el eje X y eje Y mediante el botón derecho del ratón.
- Desplazamiento en el eje Z mediante la rueda del ratón.

9.6.3 Ejemplo de ejecución de la aplicación

El visor desarrollado en este proyecto nos permite abrir archivos .ply proporcionados por Bundler y PMVS como escenario, además de posicionar otros archivos .ply en dicho escenario a partir de una imagen del escenario y otra del objeto a posicionar las cuales tengan puntos de interés en común.

Para abrir un modelo .ply como escena abriremos el visor y nos dirigiremos a Archivo -> Cargar Escena

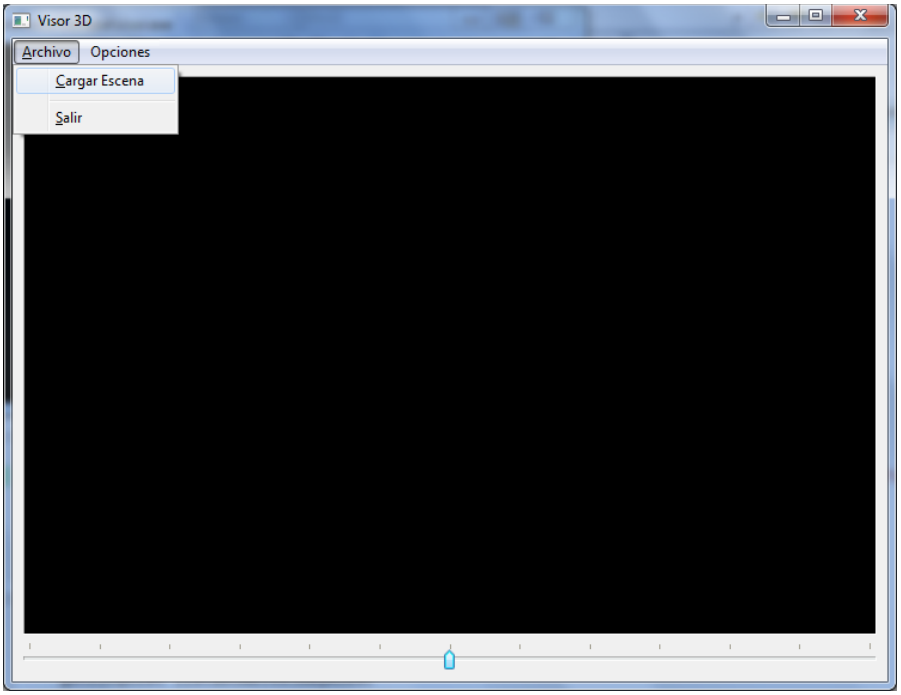


Figura 5: Ejemplo para abrir una escena.

Una vez le hayamos dado nos dirigiremos a la carpeta que contiene el archivo .ply de la escena.

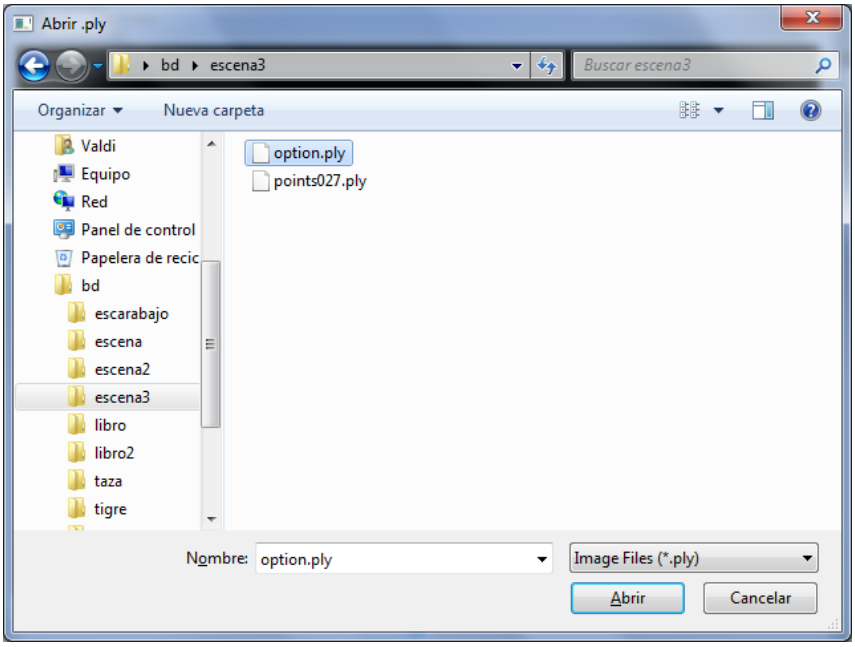


Figura 6: Ejemplo de selección de escena.

Tras seleccionar la escena se mostrara una barra de carga indicando el tiempo que falta para cargar el modelo. Dicho tiempo dependerá del número de puntos que haya en la escena.

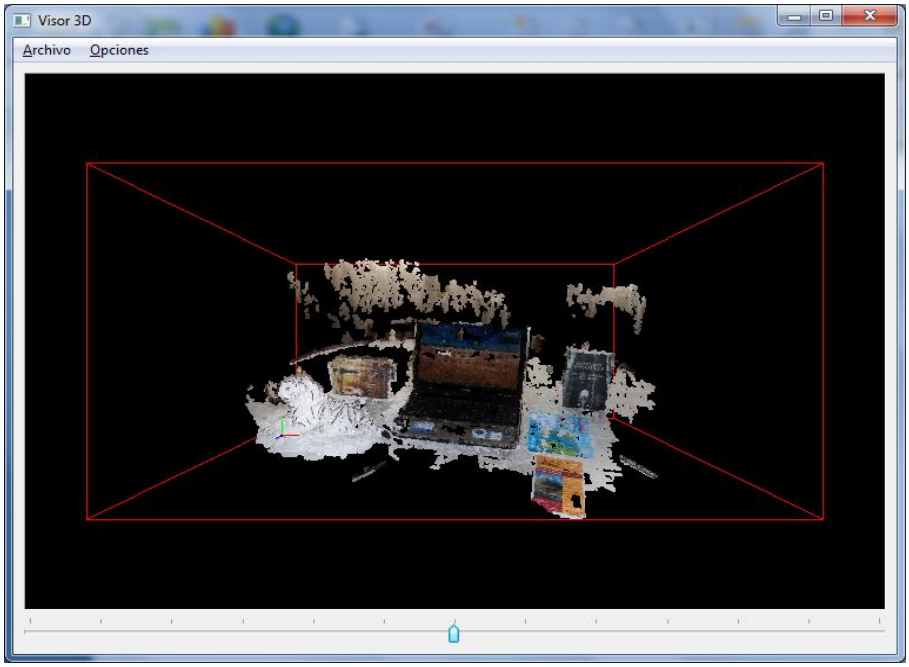


Figura 7: Ejemplo de escena cargada.

Una vez se dispone de la escena cargada podemos posicionar un objeto en ella de forma manual seleccionando una imagen de la escena y otra del objeto que sepamos de antemano que comparten puntos de interés en común o posicionar un objeto de forma automática seleccionando la carpeta de la escena y la carpeta del objeto donde se encuentran las imágenes, el problema de esta búsqueda es que es algo costosa en tiempo.

Optando por la primera opción, nos dirigiremos a Opciones -> Añadir Objeto -> Manualmente

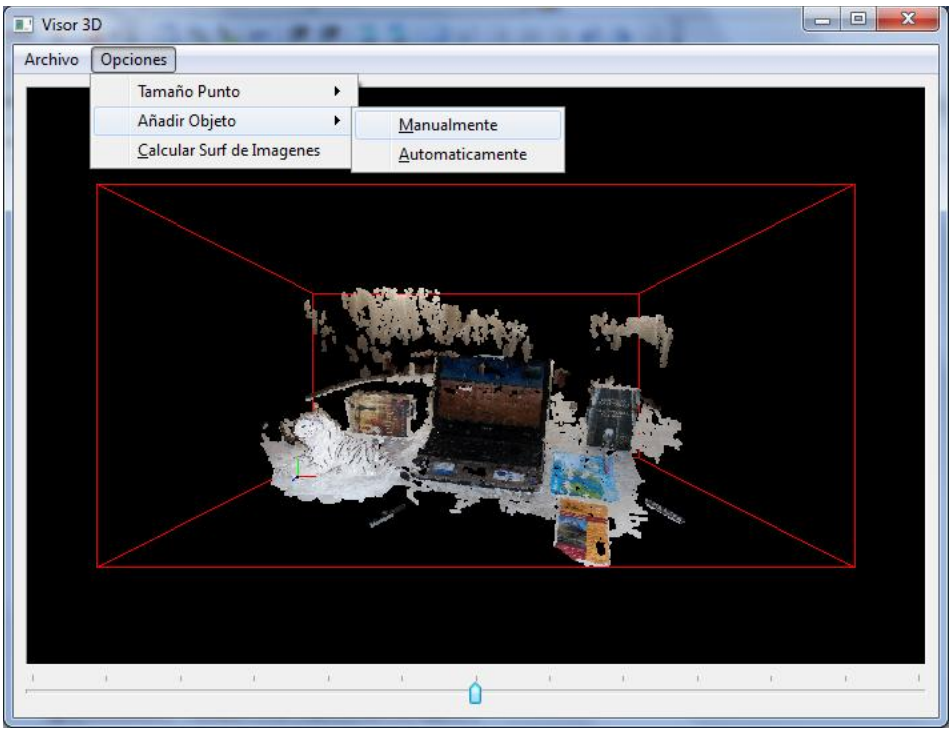


Figura 8: Ejemplo para añadir un objeto manualmente.

Tras seleccionar dicha opción se deberá seleccionar inicialmente la imagen de la escena que se quiere utilizar para posicionar el objeto y posteriormente la imagen del objeto. Ambas imágenes deben encontrarse en formato pgm para poder trabajar con ellas.

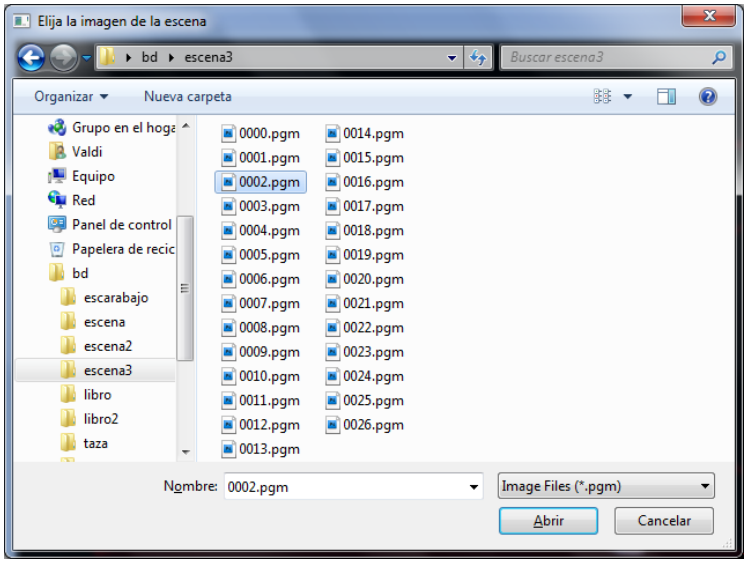


Figura 9: Ejemplo de selección de la imagen de la escena a utilizar para el posicionamiento.

En este caso como se ha mencionado previamente debemos asegurarnos de que las imágenes tienen puntos en común ya que si no sería imposible encontrar emparejamientos y con ello posicionar el objeto. A continuación se muestra un ejemplo de las imágenes seleccionadas por nosotros para este ejemplo.

Figura 10: Ejemplo de las imágenes seleccionadas para la escena y para el objeto.

Cuando se hayan seleccionado ambas imágenes el programa procederá a buscar emparejamientos, para que esto sea posible en la misma carpeta donde se encuentran las imágenes se deberán encontrar los archivos .key proporcionados por Bundler al obtener el modelo, en los cuales se encuentran los puntos de interés y descriptores SIFT de cada imagen. Además en dichas carpetas se deberá encontrar también el archivo bundle.out proporcionado también por Bundler en el cual se almacena la posición de cada cámara o imagen, ya que sin esa información no es posible posicionar el objeto correctamente.

El proceso de emparejamiento es algo costoso debido a que para este proyecto no se decidió utilizar kd-tree por lo que el tiempo puede variar mucho dependiendo de la similitud de los descriptores y del número de puntos de interés de cada imagen.

Cuando termine el proceso de emparejamiento, en caso de que se haya podido calcular correctamente la matriz fundamental se pedirá el modelo del objeto a posicionar en la escena.

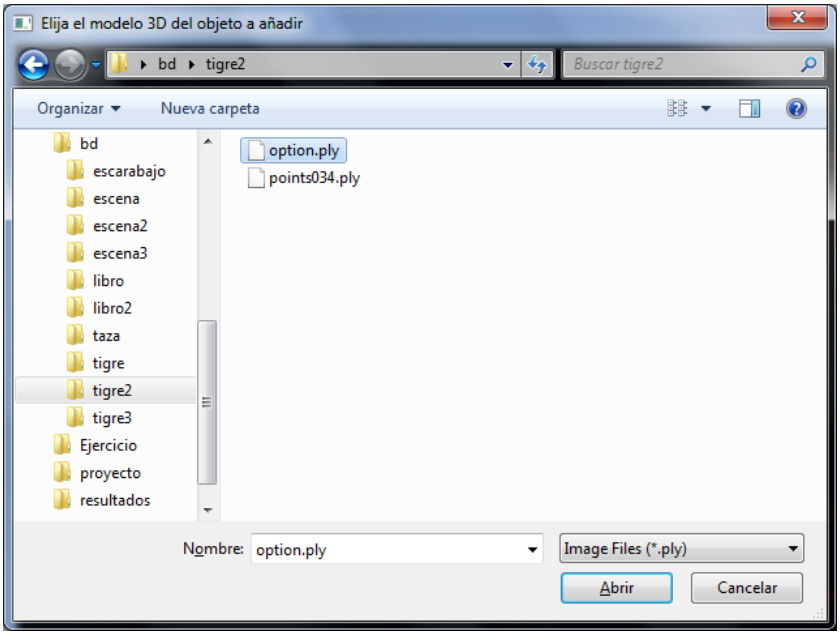


Figura 11: Ejemplo de selección del modelo del objeto.

Tras proporcionarle al visor el modelo del objeto a posicionar aparecerá en pantalla dicho modelo en la posición que se ha calculado como correcta.

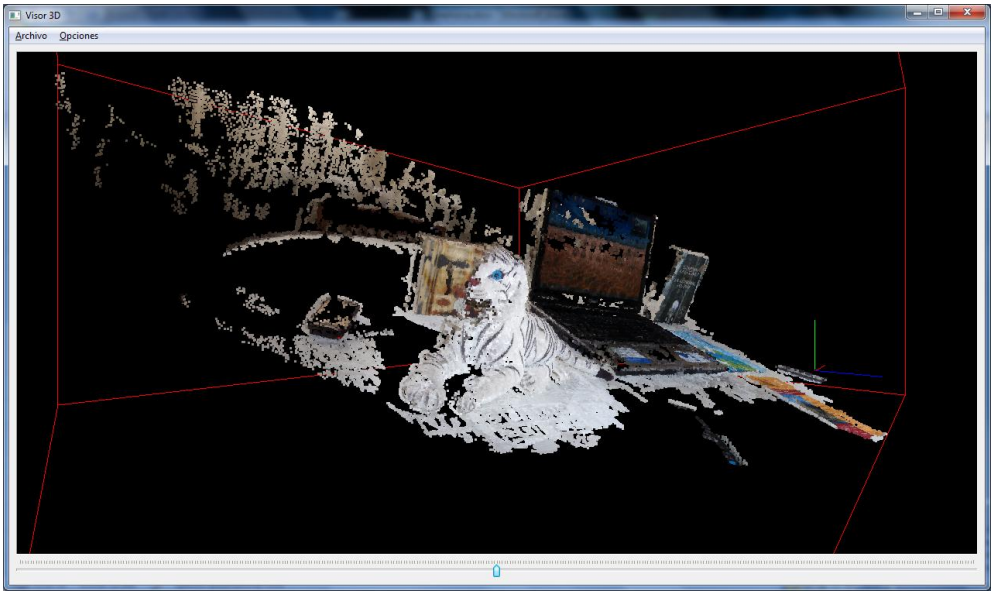


Figura 12: Modelo antes de añadir el objeto.

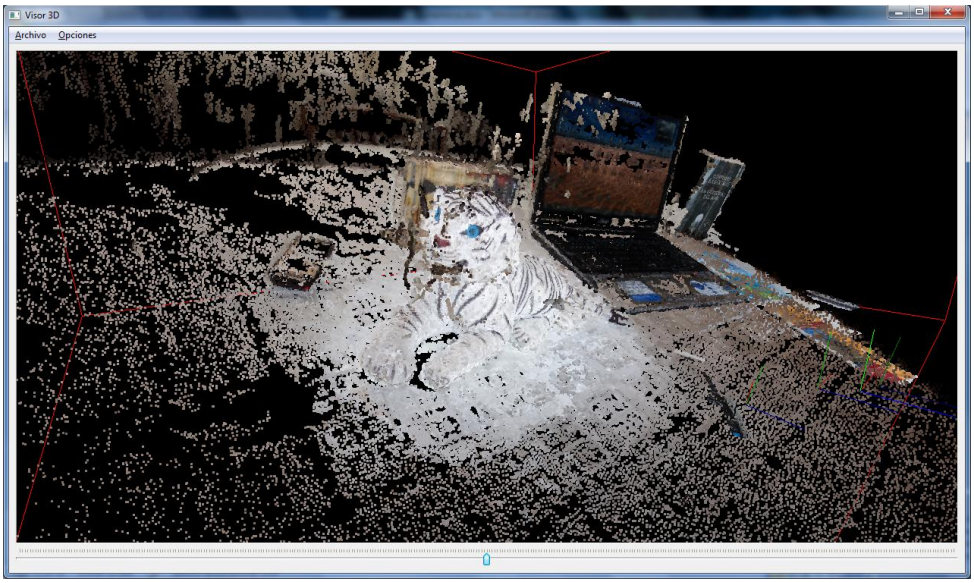


Figura 13: Modelo después de añadir el objeto.

9.7 Datasets

9.7.1 Dataset 1

Escena

Figura 14: Varias imágenes realizadas en la cafetería del C.P.S para la escena del primer dataset.

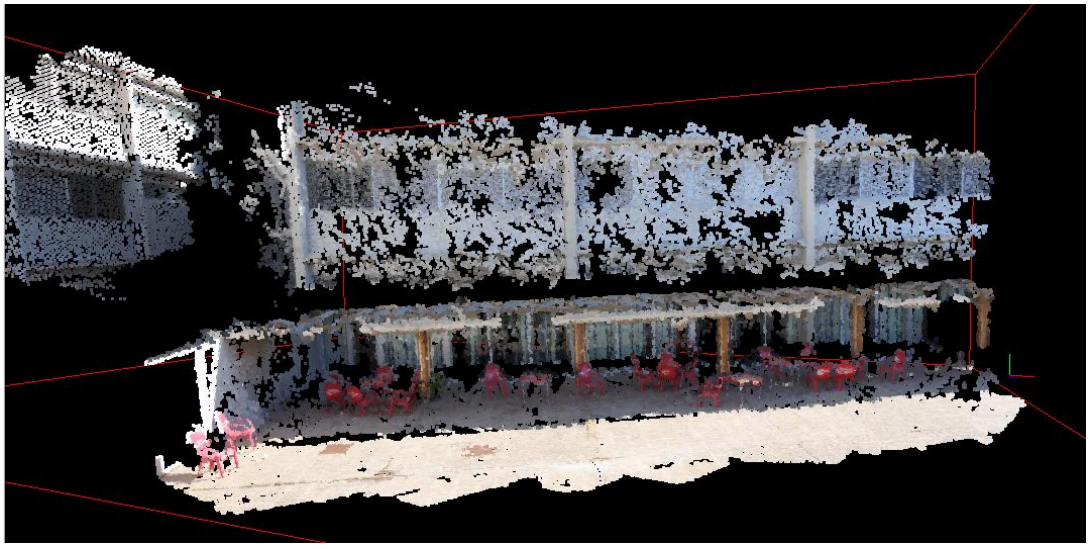


Figura 15: Reconstrucción 3D de la cafetería.

Objetos

Silla

Figura 16: (izq) Ejemplo de fotografía tomada para obtener el modelo de las sillas de la cafetería. (der) Reconstrucción 3D de la escena de la silla.

Mesa

Figura 17: Ejemplo de fotografía tomada para obtener el modelo de las mesas de la cafetería. (der) Reconstrucción 3D de la escena de la mesa.

9.7.2 Dataset 2

Escenas

Escena 1

Figura 18: Varias imágenes realizadas para obtener el modelo 3D de la escena 1 del segundo dataset.

Figura 19: Reconstrucción 3D de la escena 1 del segundo dataset.

Escena 2

Figura 20: Varias imágenes realizadas para obtener la segunda escena del segundo dataset.

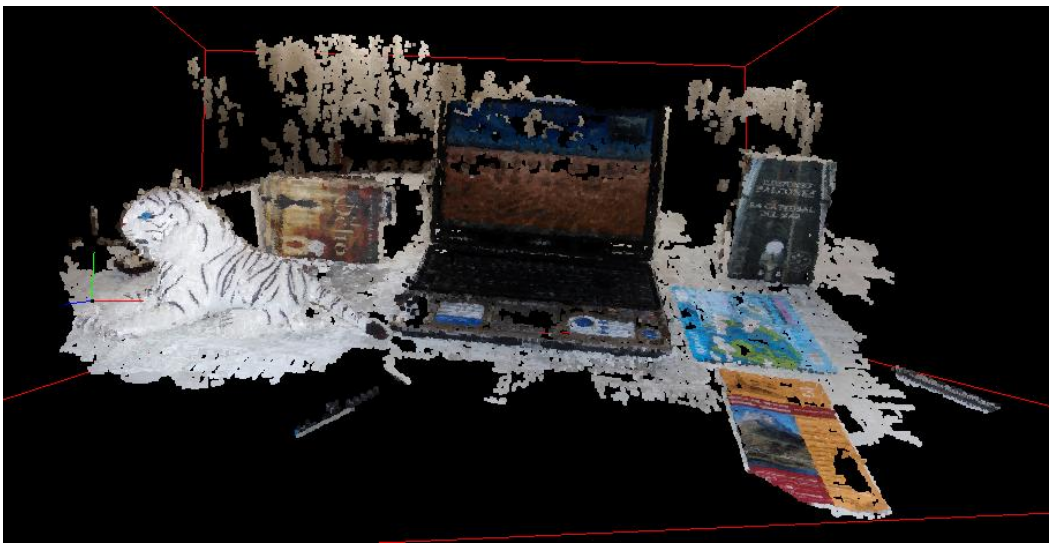


Figura 21: Reconstrucción 3D de la segunda escena.

Objetos

Libro

Figura 22: (izq) Ejemplo de foto realizada para la obtención del modelo 3D del libro. (der) Reconstrucción 3D del libro.

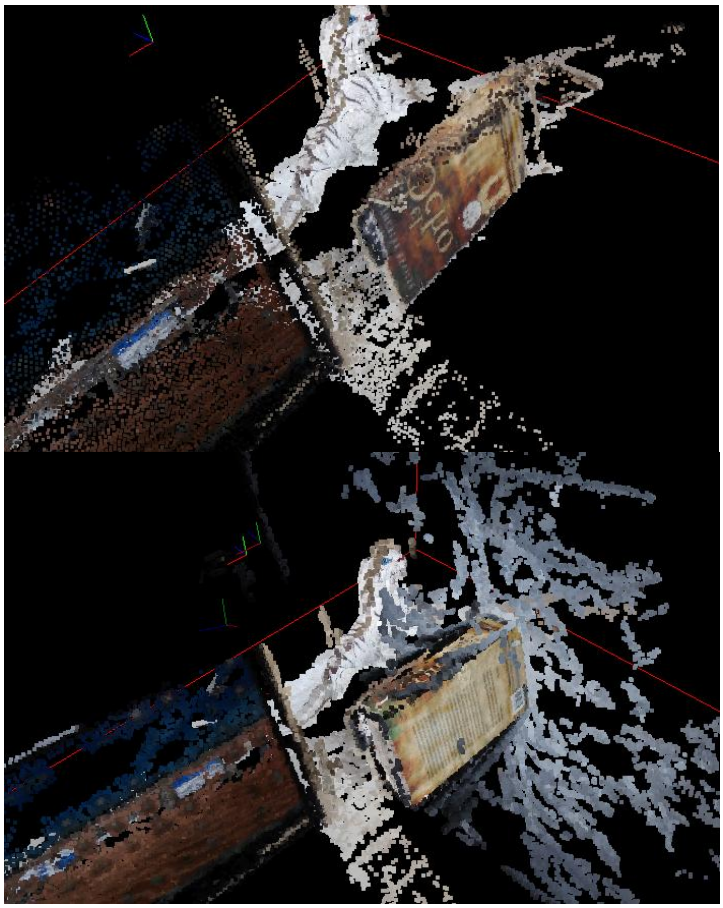


Figura 23: Ejemplo de posicionamiento del libro en la segunda escena. En ambas imágenes se visualiza la escena por detrás, en la primera antes de posicionar el libro y en la segunda una vez posicionado el libro.

Peluche

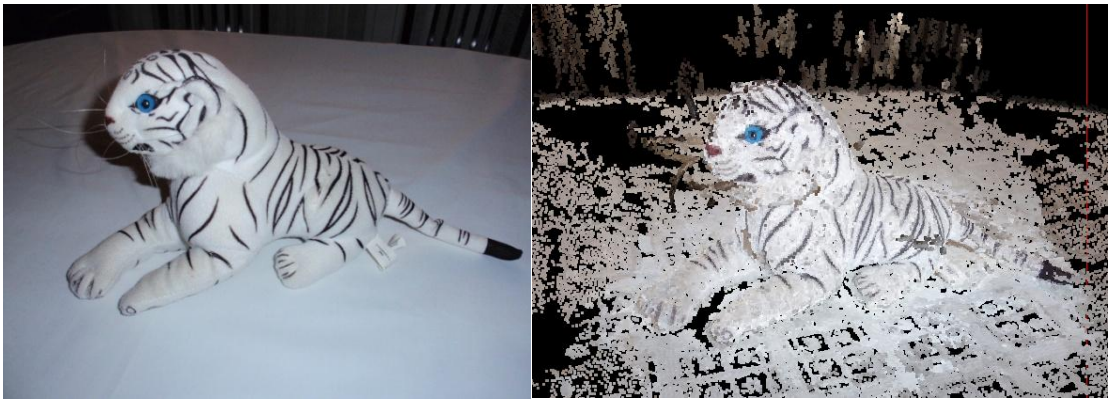


Figura 24: (izq) Ejemplo de imagen utilizada para la obtención del modelo 3D del peluche. (der) Reconstrucción 3D del peluche.

Taza

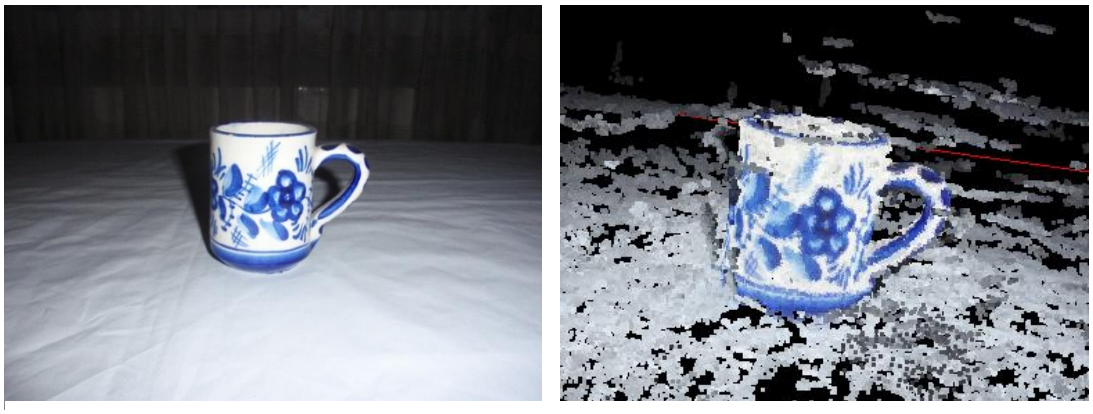


Figura 25: (izq) Ejemplo de imagen utilizada para la obtención del modelo 3D de la taza. (der) Reconstrucción 3D de la taza.

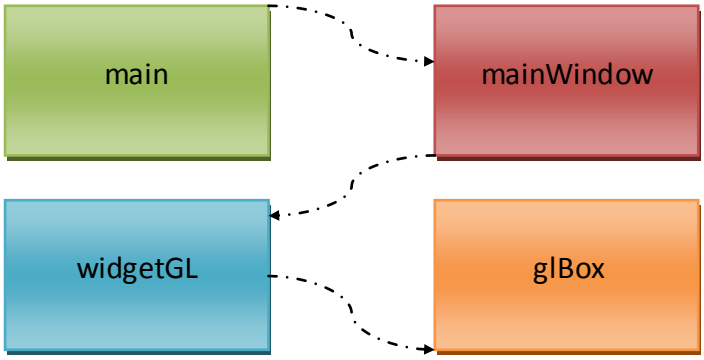
Escarabajo

Figura 26: (izq) Ejemplo de imagen utilizada para la obtención del modelo 3D del escarabajo. (der) Reconstrucción 3D del escarabajo.

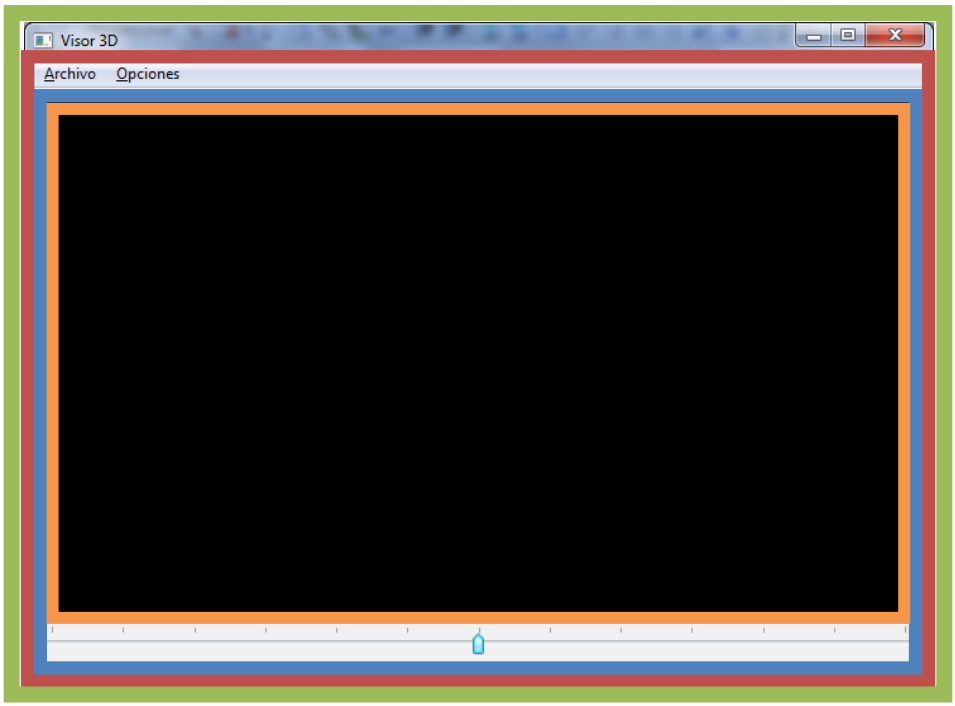
9.8 Estructura del proyecto

A continuación se procede a explicar los diferentes módulos que forman el proyecto desarrollado. Antes de proceder hay que recordar que el proyecto ha sido programado en C++ utilizando las bibliotecas Qt y OpenCV.

La aplicación está formada por cuatro módulos diferenciados:



Las flechas indican el orden de creación de las clases principales que forman el programa, además de que modulo es el encargado de crear cada clase.



En esta captura de pantalla se muestra mediante rectángulos de colores sobre que partes del programa tiene funcionalidad cada modulo.

9.8.1 main

Modulo principal desde el cual se crea y muestra la ventana principal del programa. Para ello se crea la clase *mainWindow* y después se llama a la función show().

```
...
mainWindow window;
window.setWindowTitle("Visor 3D");
window.show();
...
```

9.8.2 mainWindow

Clase de tipo QMainWindow (Clase de tipo Ventana predeterminada de Qt) desde la cual se crea la clase GLWidget, la cual se establece como widget central de la clase mainWindow. Además desde esta clase se crean y enlazan los menús superiores del programa.

```
...
//Establecemos como widget principal del programa el GLWidget

GLWidget = new widgetGL;
setCentralWidget(GLWidget);

//Creamos los menus del programa principal

crearMenu();

...
```

Procedimientos y funciones

Para facilitar la comprensión de los procedimientos y funciones se ha decidido suprimir la lista de parámetros de entrada y salida ya que muchos de ellos son de tipos específicos de las librerías Qt y OpenCV utilizadas en este proyecto. A continuación se muestran los procedimientos más importantes implementados para esta clase.

crearMenu()
Procedimiento encargado de crear y enlazar el menú superior del programa.

abrirFichero()
Procedimiento encargado de abrir el archivo .ply que se quiere utilizar como escena del programa.

anadirObjetoMan()
Procedimiento encargado de añadir y posicionar un objeto en la escena a partir de dos imágenes (una correspondiente a la escena y otra al objeto) seleccionadas manualmente por el usuario.

calcularCoincidenciasSift()
Procedimiento encargado de obtener los emparejamientos Sift de las dos imágenes.

calcularMatrizFundamental()
Procedimiento encargado de calcular la matriz fundamental a partir de los emparejamientos.

rellenarMatrizIntrinseca()

Procedimiento encargado de rellenar la matriz intrínseca de una cámara a partir de varios datos de entrada como son su distancia focal, el tamaño de la imagen, etc.

calcularMatrizEsencial()

Procedimiento encargado de calcular la matriz esencial a partir de la matriz fundamental y las matrices de parámetros intrínsecos de las cámaras.

calcularRt()

Procedimiento encargado de calcular la matriz de rotación y el vector de translación a partir de la matriz esencial.

calcularEscalaEscObj()

Procedimiento encargado de calcular la escala del modelo de la escena y del objeto.

rellenarMatrizIntrinseca()

Procedimiento encargado de rellenar cada elemento de la matriz intrínseca de una cámara a partir de unos determinados parámetros.

obtenerRtBundler()

Procedimiento encargado de obtener la matriz de rotación y el vector de translación de una cámara proporcionados por el bundler.

9.8.3 widgetGL

Clase de tipo QWidget desde la cual se establece el aspecto y la posición de los distintos elementos que componen la aplicación, como son el slider inferior al programa o el frame negro situado en el centro de la aplicación. Además desde esta clase se crea un segundo widget denominado glBox en el cual se utilizaran procedimiento de la librería openGL para mostrar y movernos por los modelos 3D.

```
...
//Se crea un frame negro y se establece el widget glBox como
//widget de ese frame
QFrame* f = new QFrame(this);
f->setFrameStyle( QFrame::Sunken | QFrame::Panel );
c = new glBox( f, "glbox");

//Creamos el slider horizontal
QSlider* z = new QSlider(Qt::Horizontal, this);
//Conectamos el slider a la función setZRotation del widget
//glBox
QObject::connect(z,SIGNAL(valueChanged(int)),c,
                 SLOT(setZRotation(int)));
//Creamos contenedores horizontales y verticales en los que
//introducir para posicionar en la aplicación los elementos
//previamente creados
QHBoxLayout* hlayout = new QHBoxLayout(this);
QVBoxLayout* vlayout = new QVBoxLayout();
QHBoxLayout* flayout = new QHBoxLayout(f);

flayout->setMargin(0);
flayout->addWidget( c, 1 );
vlayout->addWidget( f, 1 );
vlayout->addWidget( z );
hlayout->addLayout( vlayout );
...
```

9.8.4 glBox

Clase de tipo QGLWidget (Clase predeterminada de Qt para definir widgets de tipo OpenGL) en la cual se encuentran las funciones y procedimientos necesarios para visualizar los modelos 3D además de los procedimientos y funciones para girar y desplazar el modelo.

Se ha decidido no mencionar ni describir los algoritmos implementados para esta clase ya que la gran mayoría de ellos son específicos para tratar el modulo OpenGL de la librería Qt o para trabajar con eventos de ratón, teclado etc. de la librería Qt aportando así poca información sobre el proyecto.

9.9 Bundler

Desde 1980 empezaron a surgir los primeros indicios y técnicas para obtener las posiciones de las cámaras a partir de un conjunto de fotografías, desarrollándose en 1999 por Triggs una técnica denominada *Bundle Adjustement*. Esta técnica poco a poco se fue haciendo sitio desde entonces, hasta convertirse en un estándar para la reconstrucción 3D. Bundler se basa en esta técnica pero incorpora algunas modificaciones para hacerlo más robusto en caso de recibir imágenes realizadas con diferentes cámaras.

Inicialmente Bundler obtiene los puntos de interés de cada imagen, para ello hace uso del algoritmo SIFT (Scale-invariant feature transform), el cual le proporciona un descriptor de 128 bits por cada punto (una imagen típica contiene entorno a varios miles de puntos SIFT). Los descriptor SIFT se basan en histogramas de orientación del gradiente, consiguiendo con ello ser invariantes a transformaciones como pueden ser de rotación, translación, de escala e incluso parcialmente invariante a cambios de iluminación.

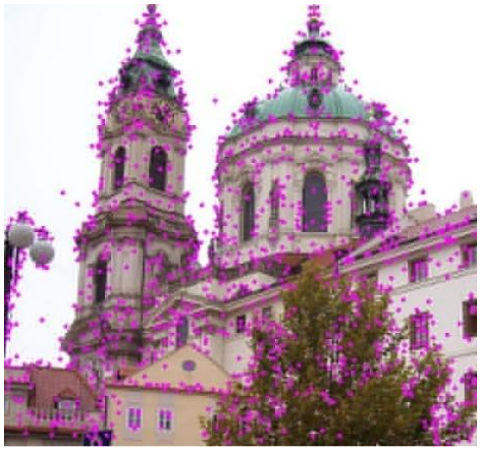


Figura 27: Ejemplo de puntos característicos de una imagen utilizando SIFT. [15]

Una vez Bundler ha obtenido los puntos y descriptores SIFT de las imágenes, procede a emparejar los diferentes puntos de cada imagen comparando los descriptores de una con los de la otra, calculando para ello la distancia euclidea entre estos mediante la fórmula:

$$\mathrm{d}(\mathbf{p}, \mathbf{q}) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2}$$

Para mejorar el tiempo de cálculo emplea la implementación de un algoritmo de aproximación al vecino más cercano (ANN - approximate nearest neighbors) mediante kd-tree (Arya - 1998).

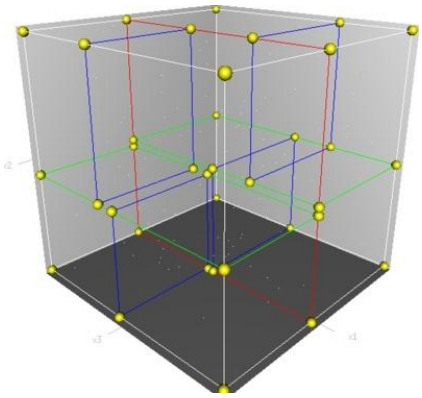


Figura 28: Ejemplo de un kd-tree en 3 dimensiones. [15]

Para emparejar los puntos entre una imagen I y otra J, se crea un kd-tree de los descriptores de cada punto de la imagen J. Entonces se procede para cada punto de I a buscar el vecino más próximo en J usando el kd-tree previamente creado. Por razones de eficiencia se limita a 200 el número máximo de núcleos a visitar en el árbol. Bundler en lugar de eliminar los falsos emparejamientos por un error máximo o threshold en la distancia entre dos descriptores, utiliza el método de Lowe (2004). Este método consiste en que para descriptor de I se busca los dos mejores

emparejamientos de J, si la distancia para cada uno es d_1 y d_2, aceptamos el emparejamiento si $d_1/d_2 < 0.6$. En caso de que algún punto de I se empareje con un punto de J que ya había sido previamente emparejado, se eliminan todos los emparejamientos relacionados con el punto de J.

Una vez tenemos los emparejamientos entre ambas imágenes, Bundler procede a calcular la matriz fundamental mediante el algoritmo de los 8 puntos utilizando previamente RANSAC. Con RANSAC conseguimos eliminar posibles errores de emparejamiento.

El algoritmo de RANSAC funciona básicamente de la siguiente manera:

- Seleccionar aleatoriamente dos puntos, los cuales definen una recta.
- Comprobar cuántos puntos pertenecen a esa recta, calculando para ello la distancia de ellos a la recta, si esta distancia es inferior a un valor previamente fijado se considerara a ese punto como Inlier en caso contrario se denominara Outlier.
- Se repite la selección aleatoria un número determinado de veces. La recta que posea un mayor número de puntos será considerada como la recta robusta.

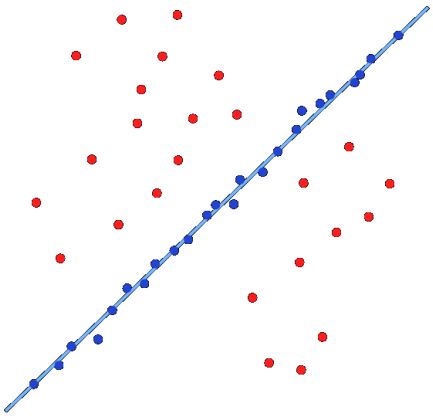


Figura 29: Ejemplo de Inlier y Outlier en RANSAC. [1]

Bundler utiliza un valor del 0.6% de las dimensiones máximas de la imagen como threshold (distancia máxima de un pixel a la recta).

Una vez hemos calculado la matriz Fundamental, esta es refinada mediante el algoritmo de Levenberg-Marquadt minimizando con ello los errores de todos los inliers de la matriz Fundamental. Posteriormente se eliminan los emparejamientos que son considerados outliers y se comprueba que el numero de emparejamientos restantes es superior a veinte.

El siguiente paso consiste en organizar los emparejamientos en vías, donde una vía es un conjunto de puntos clave que conectan múltiples imágenes. Si una vía contiene más de un punto clave en la misma imagen, esta se denomina inconsistente. El objetivo es mantener vías consistentes que contengan al menos dos puntos clave para la siguiente fase de reconstrucción. Una vez que se han organizado las vías ya podemos construir un grafico de foto conectividad, en el cual cada imagen es un nodo y una arista existe entre un par de imágenes con emparejamientos.

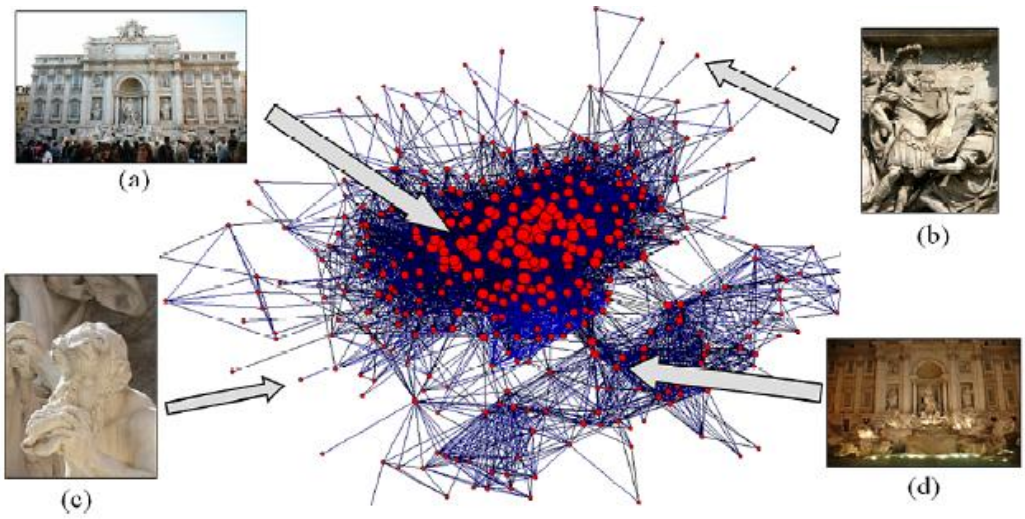


Figura 30: Grafico de foto conectividad. Este grafico contiene un nodo por cada imagen de un conjunto de fotografías de la Fuente de Trevi, con una arista uniendo dos nodos que comparten emparejamientos. El tamaño de cada nodo es proporcional a su grado. Existen dos núcleos claros, uno corresponde a fotografías de día (a) y otras de noche (d). Las imágenes menos usuales aparecen alejadas del centro de los núcleos (b) y (c). [1]

A continuación recuperamos los parámetros de la cámara (rotación, translación y distancia focal) para cada imagen y una localización 3D para cada vía. Los parámetros recuperados han de ser consistentes, de forma que la suma de distancias entre las proyecciones de cada vía y sus características de la imagen sean mínimas. Este problema de minimización puede ser formulado como un problema no lineal de mínimos cuadrados y resuelto usando bundle adjustment. Los algoritmos para resolver este tipo de problemas no lineales como es el caso de Nocedal and Wright solo garantizan encontrar mínimos locales, por lo que es importante proporcionarle una buena aproximación inicial de los parámetros. En lugar de estimar los parámetros para todas las cámaras y vías a la vez, se hace una aproximación incremental, añadiendo una cámara cada vez.

Se empieza haciendo una estimación de los parámetros de un único par de cámaras. Este par de cámaras debe tener un gran número de emparejamientos, pero no pueden ser modelados por una sencilla homografía. Es decir, elegiremos el par de cámaras que tengan un mayor número de emparejamientos pero evitando que sean cámaras coincidentes. En particular, se busca una homografía entre cada par de imágenes emparejadas usando RANSAC con un threshold de 0.4% de las dimensiones máximas de la imagen y se almacena el porcentaje de emparejamientos que son inliers para dicha homografía. Se selecciona el par de imágenes con menor porcentaje de inliers para la homografía recuperada, pero con al menos cien emparejamientos. Los parámetros de la cámara para este par son estimados usando la implementación de los cinco puntos de Nistér, entonces las vías visibles en ambas imágenes son triangularizadas. Finalmente se hace un bundle adjustment partiendo de esta inicialización.

A continuación se añade otra cámara a la optimización. Se selecciona aquella que contiene un mayor número de vías cuya localización 3D ha sido ya estimada. Los parámetros extrínsecos de la nueva cámara son inicializados usando una transformación linear directa mediante la técnica de Hartley y Zisserman dentro de un procedimiento RANSAC. Con esta técnica además de estimar los parámetros de rotación y translación de la cámara, también obtenemos una matriz K que puede ser

utilizada como una estimación de los parámetros intrínsecos. Con la matriz K calculada y la distancia focal de la información EXIF de la imagen se inicializa la distancia focal de la nueva cámara. Empezando con estos parámetros iníciales, procedemos a realizar un bundle adjustment, permitiendo cambios únicamente en la nueva cámara y en los puntos que esta observa, el resto del modelo permanece fijo.

Los puntos observados por la nueva cámara son añadidos al modelo si al menos son vistos también por otra cámara ya recuperada del modelo y si la triangularización de dichos puntos proporcionan una buena estimación de su localización. Una vez añadidos se realiza un bundle adjustment para refinar el modelo completo. Este procedimiento se repite para cada cámara hasta que no quedan cámaras que visualicen suficientes puntos 3D como para ser reconstruidas.