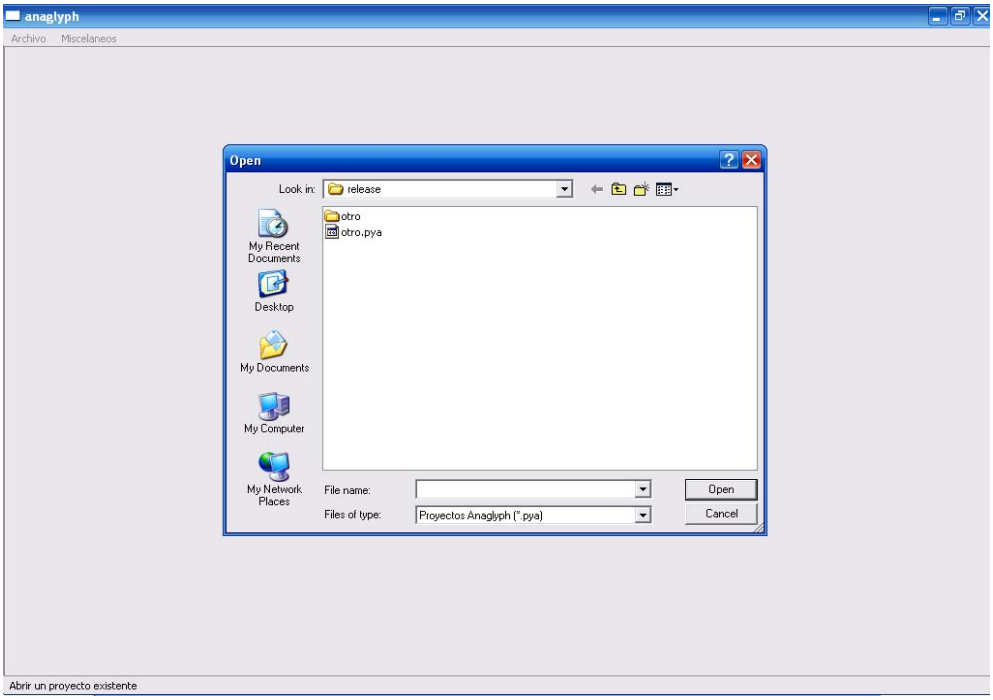

Figura Nº 26. Caja de dialogo: Abrir proyecto

En estado de un proyecto activo, se tiene la posibilidad de ajustar el sistema de coordenadas de referencia y/o el punto homologo, asignar los parámetros de cálculo, visualizar los datos registrados (si el proyecto es antiguo) y visualizar el anaglifo, todo esto, desde los menús mostrados anteriormente. La **figura Nº 27** muestra la aplicación desde un escenario de proyecto activo.

El proceso de asignar los parámetros comienza al seleccionar "Asignar parámetros" en la barra de menú, específicamente en el menú "Misceláneos" o en el menú de contexto desplegado sobre el visor fotográfico. Se visualiza una caja de dialogo donde se puede especificar cada uno de los parámetros a utilizar en el calculo de las coordenadas (**Figura Nº 28**).

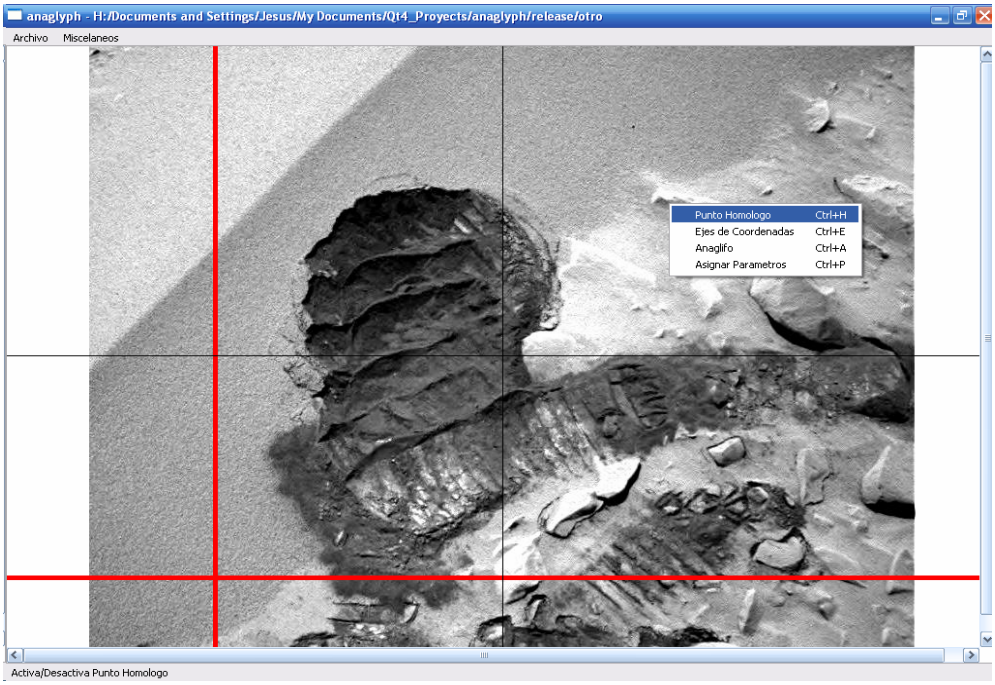

Figura Nº 27. Vista desde un proyecto activo

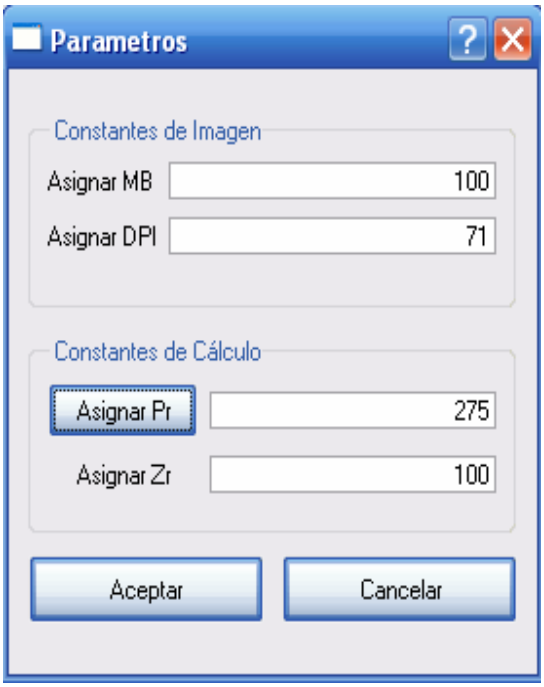

Figura Nº 28. Caja de dialogo: Asignar parámetros

Hay un parámetro que requiere de especial atención y es el que lleva por nombre "Pr", al presionar sobre el botón "Asignar Pr" en la ventana "Asignar parámetros", se despliega otra caja de dialogo con 2 visores (uno en cada lado), en el visor de la izquierda se muestra la fotografía izquierda y en el visor derecho se muestra la fotografía derecha, ambos con su respectivo sistema de coordenadas con origen en la parte media de la fotografía donde se posan; el objetivo, localizar el punto central (origen del sistema de coordenadas) de la fotografía izquierda en la fotografía derecha, moviendo el eje vertical de la fotografía derecha.

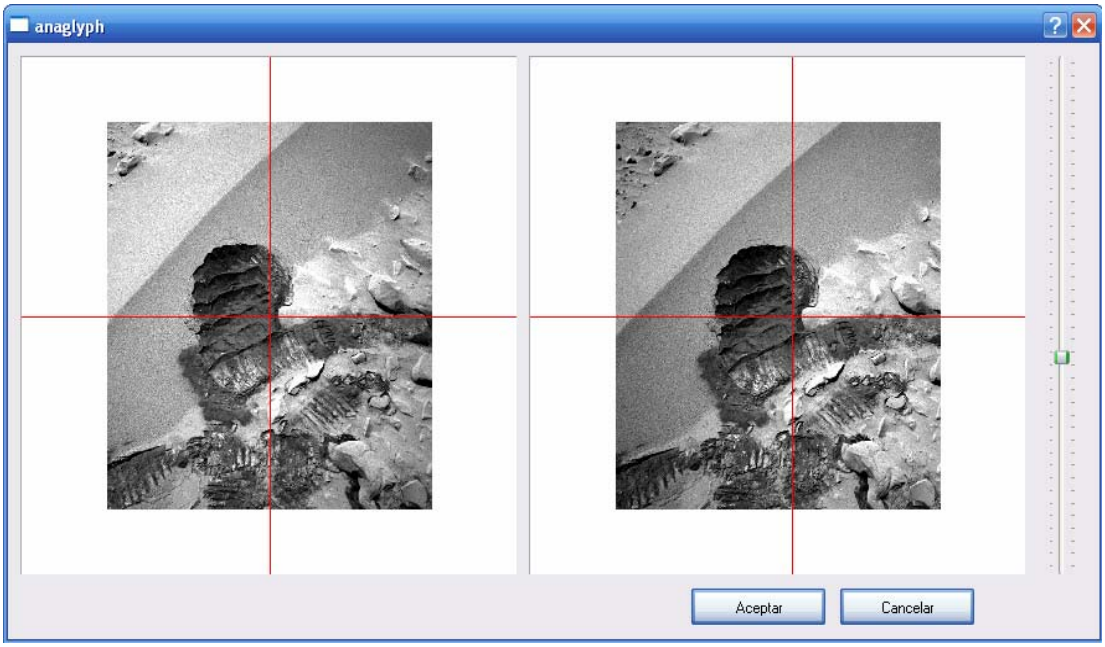

Figura Nº 29. Caja de dialogo Asignar Pr

Si el Operador selecciona "Anaglifo" desde el menú de contexto bajo un proyecto activo, se abre una nueva ventana, en cuya parte central se encuentra el visor anaglifo con la imagen estereoscópica ya cargada, lista para el proceso de recolección de coordenadas y en el costado derecho un control barra deslizadora (slider) que puede ser manipulado con el scroll del Mouse para variar el grado de solapamiento o separación horizontal (Figura Nº 30), Una vez ajustado el punto homologo el

Operador puede registrar las coordenadas accediendo al menú de contexto y seleccionando “Datos Recogidos”, este despliega una nueva ventana donde se pueden añadir nuevas coordenadas, eliminar una coordenada en especial o eliminar todas las coordenadas (**Figura Nº 31**).

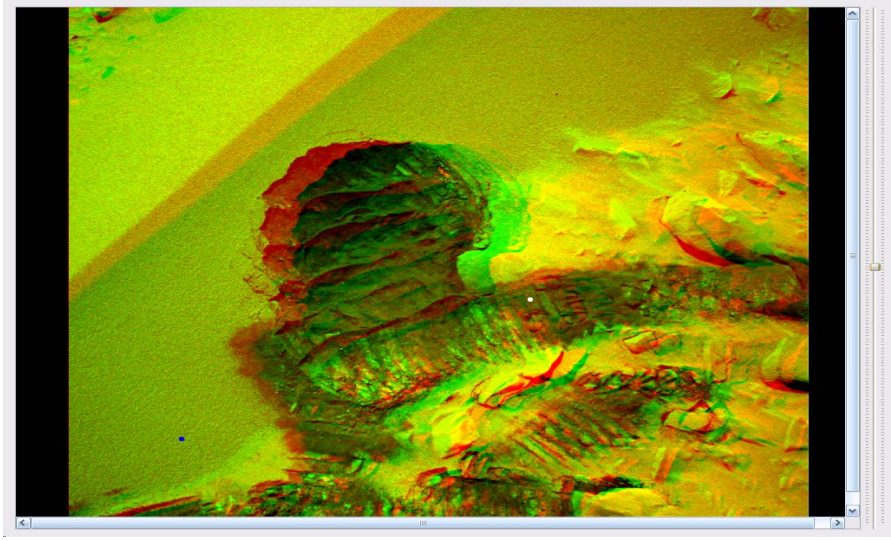

Figura Nº 30. Modo anaglifo

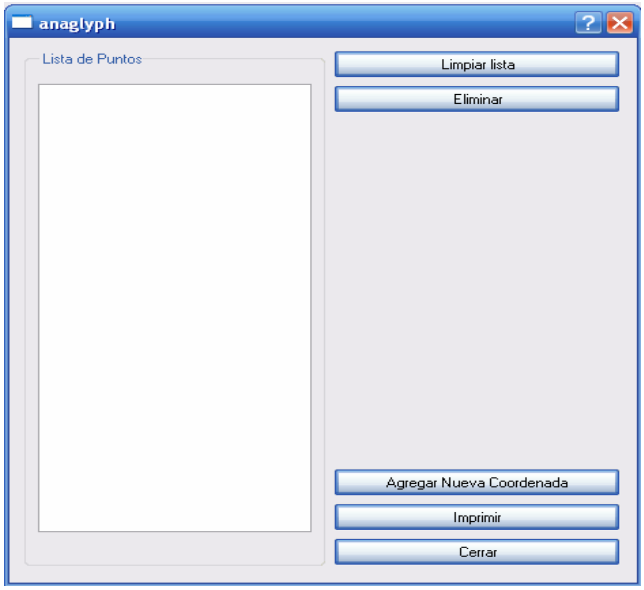

Figura Nº 31. Caja de dialogo: Datos Recogidos

CAPITULO 4
PRUEBAS

En este capitulo se especifican las pruebas realizadas al sistema como la ejecución de pruebas de caja blanca y de caja negra.

4.1 Pruebas de Caja negra

Se refiere a las pruebas que se llevan a cabo sobre la interfaz del software. Una prueba de caja negra examina algunos aspectos del modelo funcional fundamental del sistema sin tener mucho en cuenta la estructura lógica interna del software. El objetivo de estas pruebas consiste en la validación de las entradas y verificación de las opciones de salida.

1. **Creación de un Proyecto:** Hace referencia a los requerimientos funcionales R1 y R2. Se dispara una vez que el operador presiona "Nuevo" en el ítem "Archivo" de la barra de Menú en la ventana principal. Este evento muestra una caja de dialogo donde se pueden seleccionar cada una de las fotografías y asignar un nombre al proyecto. Si una de las fotografías no ha sido asignada al momento de presionar el botón "Aceptar" se muestra una ventana de aviso con el mensaje "Se deben seleccionar ambas fotografías" (**Figura Nº 32**).

   De igual manera sucede si el operador no asigna un nombre al proyecto se muestra una ventana de aviso con el mensaje "Debe asignarse un nombre al proyecto" (**Figura Nº 33**). El escenario de éxito implica que el operador halla asignado ambas fotografías y el nombre al proyecto, momentos posteriores al haber presionado el botón "Aceptar", se carga la fotografía izquierda en la ventana principal.

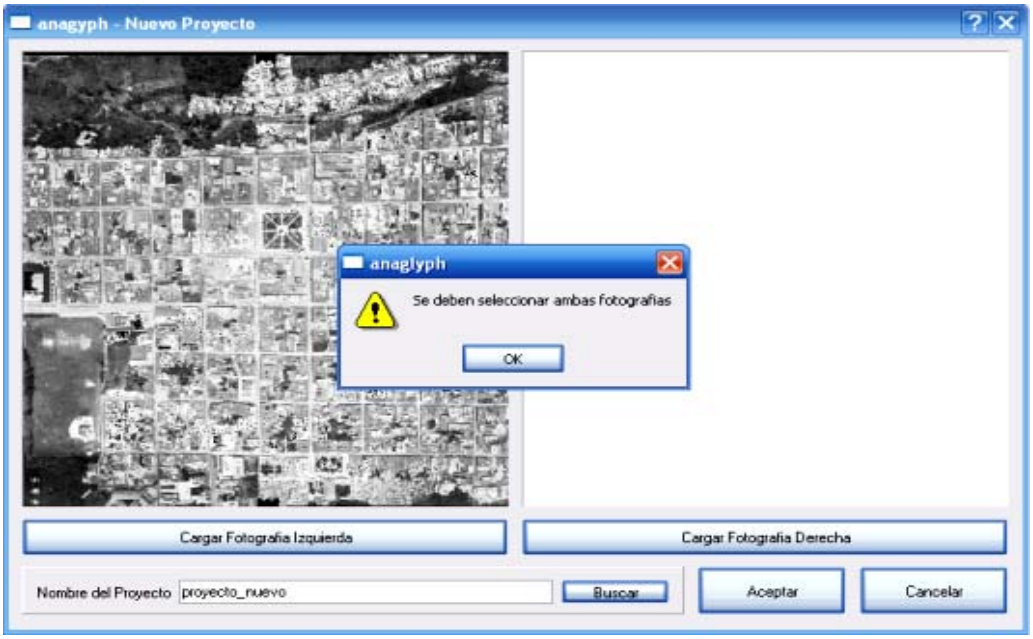

Figura Nº 32. Nuevo proyecto, error al no seleccionar una fotografía

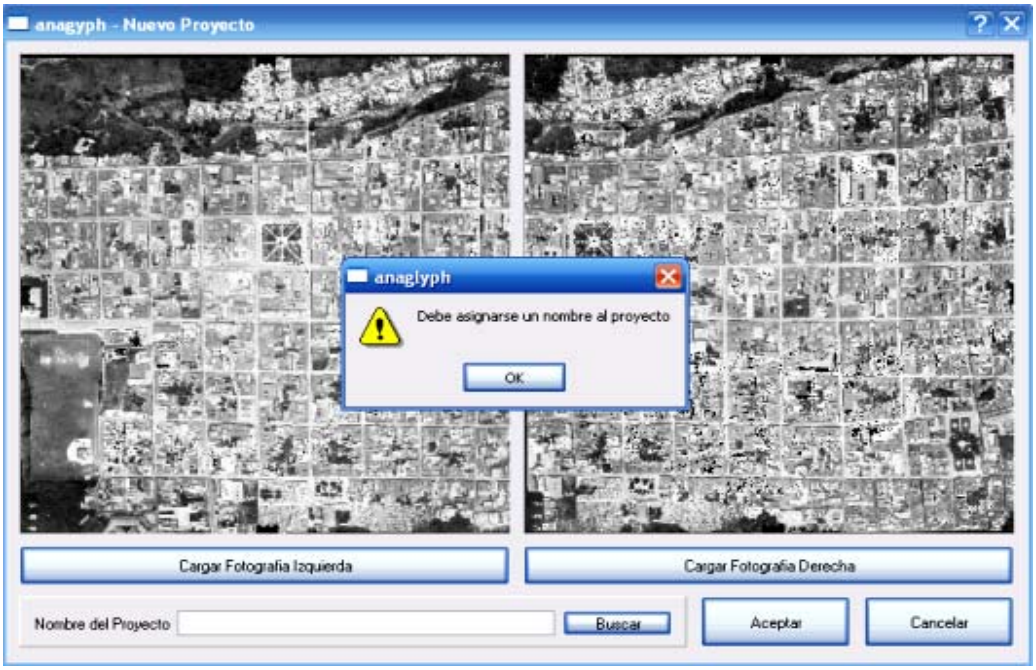

Figura Nº 33. Nuevo Proyecto, error al no asignar un nombre al proyecto

2. **Asignar parámetros:** Hace referencia al requerimiento funcional R3. se dispara una vez que el operador presiona sobre el ítem "Asignar Parámetros" del menú "Misceláneos" de la barra de menú o en el menú de contexto del visor fotográfico. Este evento muestra una caja de dialogo, donde el operador puede asignar cada uno de los parámetros que son necesarios, para el calculo de las coordenadas. Si se trata de acceder al ítem "Asignar Parámetros" sin tener un proyecto activo el sistema muestra una ventana de aviso con el mensaje "Se debe abrir un proyecto con anterioridad" (**Figura Nº 34).**

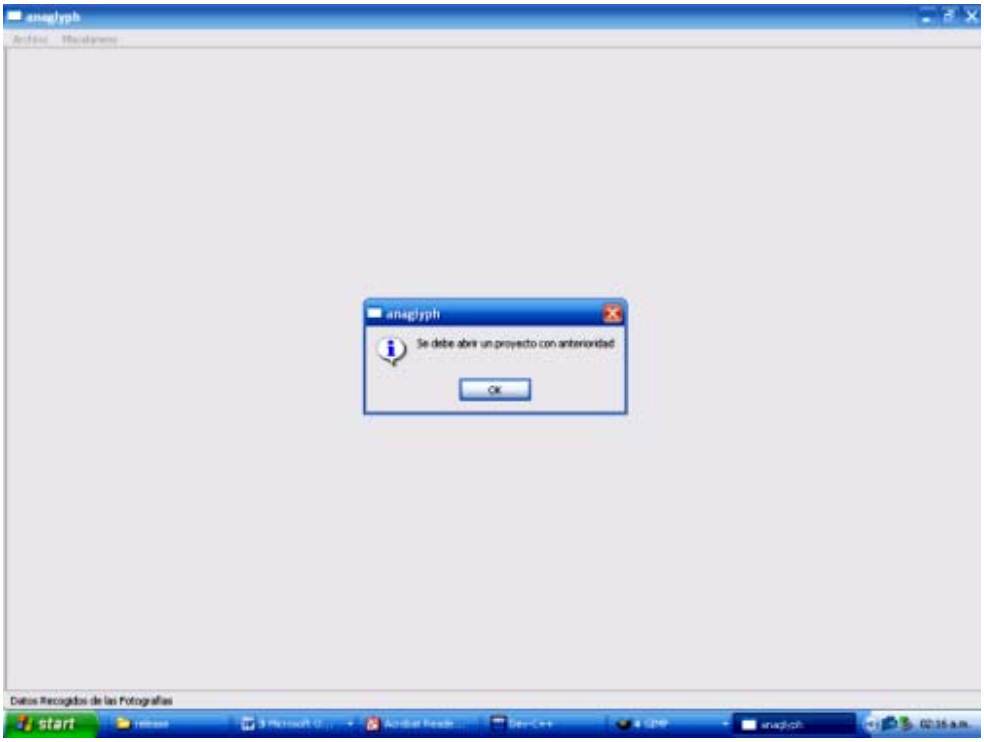

Figura Nº 34. Asignar Parámetros, error no hay proyecto activo

Si el operador olvida asignar uno de los parámetros se muestra una ventana de aviso con el mensaje "Se deben asignar todos los parámetros" (**Figura Nº 35).**

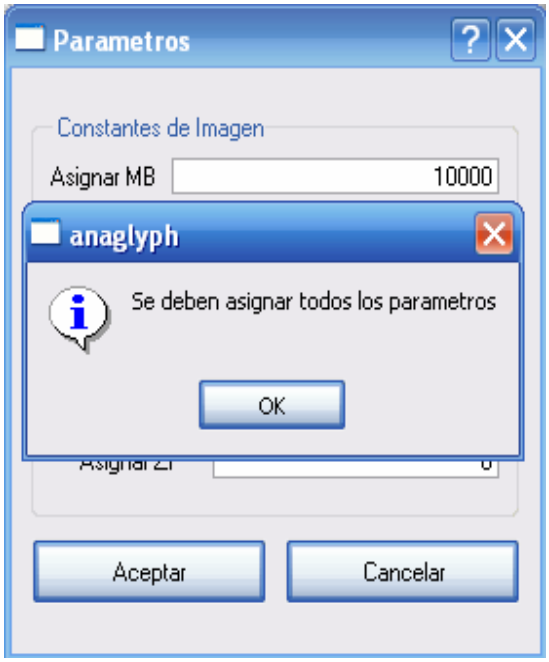

Figura Nº 36. Asignar parámetros, error parámetros faltantes

3. **Recolectar Datos:** Hace referencia al requerimiento funcional R7. Se dispara una vez que el operador presiona sobre el ítem "Datos Recogidos" del menú "Misceláneos" de la barra de menú o en el menú de contexto del visor anaglifo. Este evento muestra una caja de dialogo, donde el operador puede recolectar las coordenadas del punto homologo actual. Si se trata de acceder al ítem "Asignar Parámetros" sin tener un proyecto activo el sistema muestra una ventana de aviso con el mensaje "Se debe abrir un proyecto con anterioridad" (**Figura Nº 34).**

4.2 Pruebas de Caja Blanca

Las pruebas de caja blanca se basan en el estudio minucioso de los detalles procedimentales.

4.2.1 Ambiente de Prueba

Las pruebas fueron hechas bajo una computadora con arquitectura de 32 bits y 2.2GHz en frecuencia de reloj, 512 de RAM y un monitor de 15".

4.2.2 Experimento

Se uso el par de fotografías mostradas en la **figura Nº 37,** estas fotografías fueron tomadas del *web site* de la nasa y representan una superficie del planeta Marte.

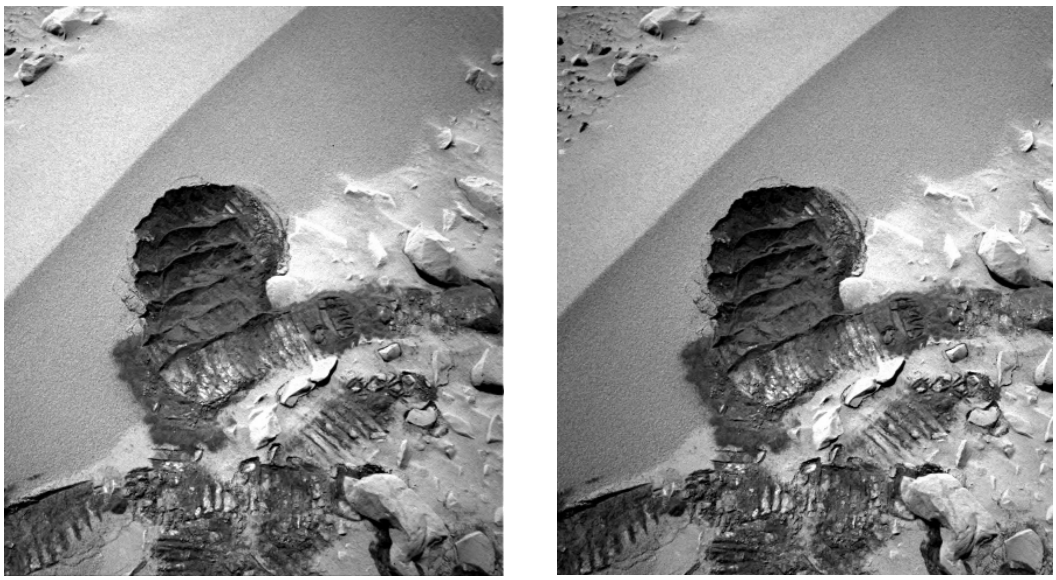

Figura Nº 37. Par Estereoscópico

Para calcular las coordenadas tridimensionales dentro de la fotografía, se necesitan los siguientes parámetros:

DPI (Puntos por pulgada) = 71
MB (Constante de escala) = 50
Zr (Altura relativa) = 3 mts

En la **figura Nº 38** se muestran los puntos (en blanco) en los cuales se basaran las coordenadas:

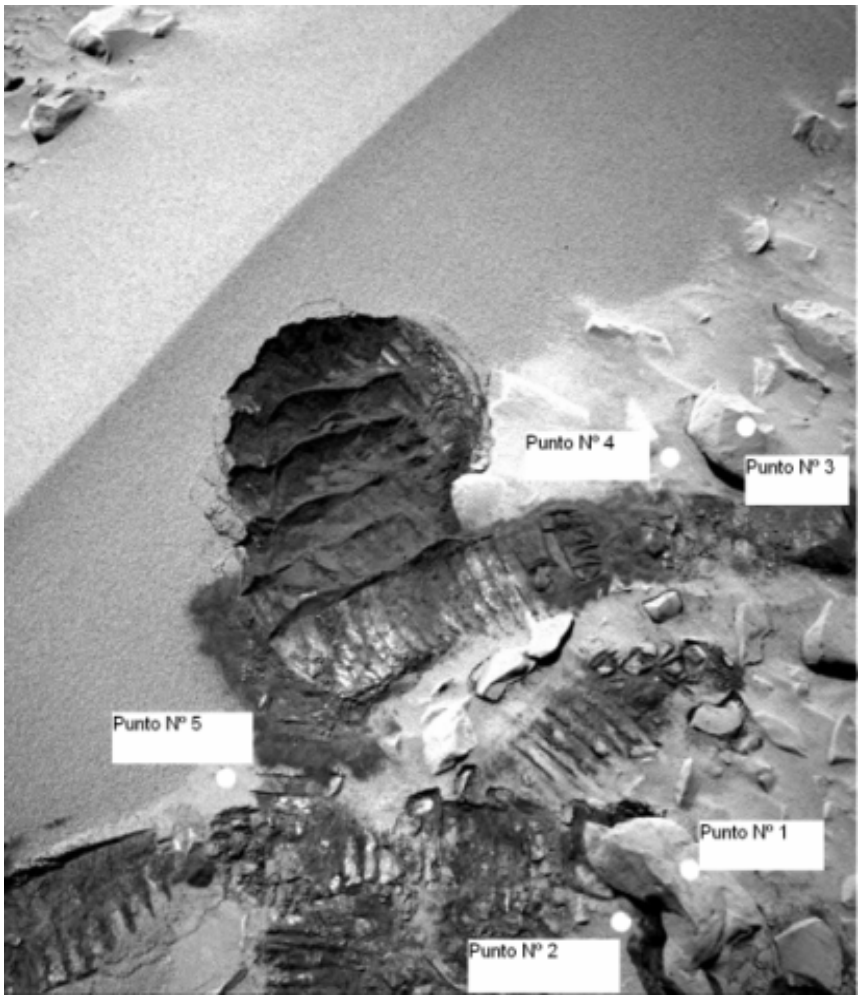

Figura Nº 38. Fotografía con los puntos objeto de estudio en blanco

Se obtuvo Pr (Paralaje relativo) a través del sistema, en el modulo de asignar parámetros mostrada en la **figura Nº 39.**

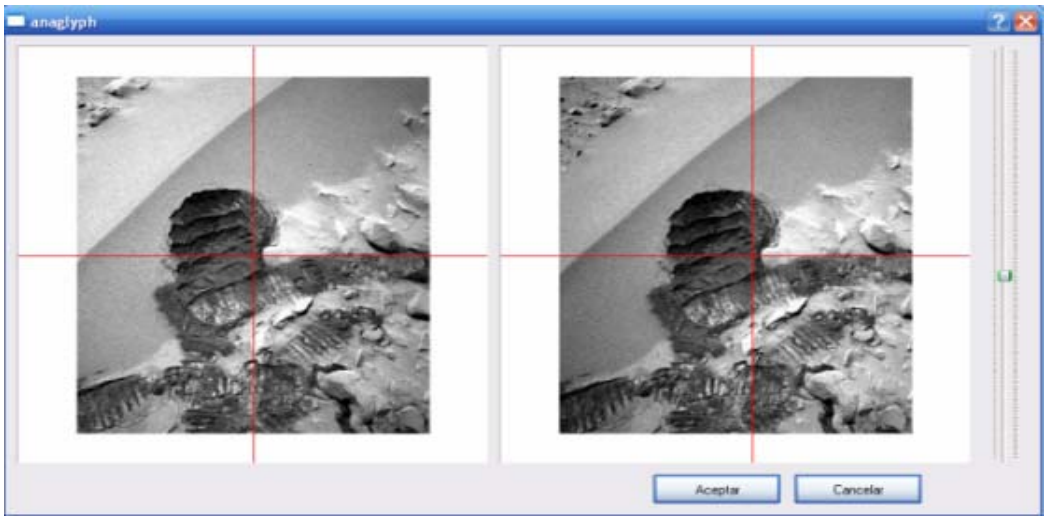

Figura Nº 39. Obtención del parámetro Pr.

Pr = 23 pixels.

El origen del sistema de coordenadas es mostrado en la **figura Nº 40,** el sistema es denotado con los ejes en rojo.

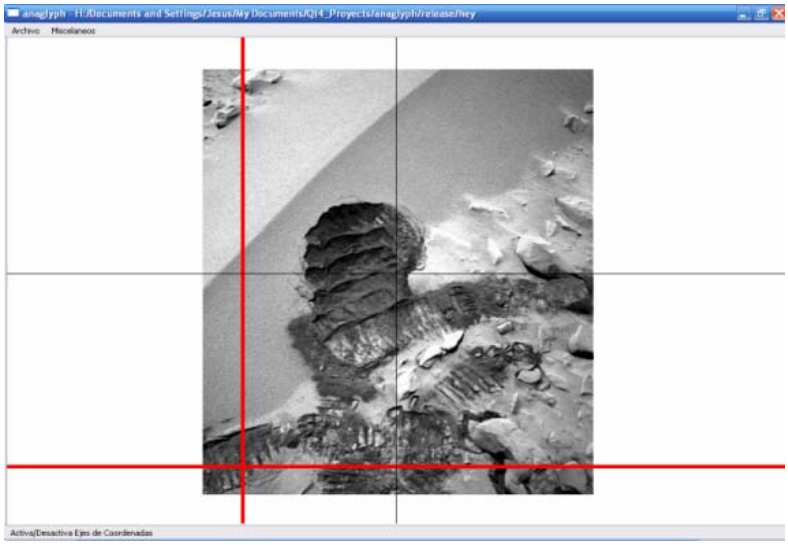

Figura Nº 40. Origen del sistema de coordenadas

La **tabla Nº 5** muestra las coordenadas obtenidas del sistema:

Punto 1	(6.35 , 0.64, 4.19) metros
Punto 2	(5.58, 0.03, 4.09) metros
Punto 3	(7,22, 5.2231, 0) metros
Punto 4	(6.68986, 5.08, 0) metros
Punto 5	(1.77085, 1.60, 7.11) metros

Tabla Nº 5. Coordenadas obtenidas del anaglifo

CAPITULO 5
CONCLUSIONES Y RECOMENDACIONES

5.1 Conclusiones

- El sistema desarrollado permitirá una interpretación fotográfica de terrenos bastante sencilla, evitando el uso de aparatos mecánicos engorrosos y difíciles de manipular.

- La función del sistema desarrollado solo se limita a interpretación, dada la naturaleza de la recolección de coordenadas, los datos obtenidos de dicho proceso son aproximaciones que dependen de la capacidad del operador para obtener la visión tridimensional a través del anaglifo, como también del parámetro Pr (paralaje relativo), que requiere una participación especial del operador para obtenerla.

- La portabilidad del sistema y el desarrollo en multiplataforma dará la posibilidad de ejecutar la aplicación en cualquier sistema operativo con solo compilar el código fuente.

- La utilización de herramientas estándares permitirá el fácil desarrollo de versiones posteriores, actualizaciones o adición de nuevos módulos dentro del sistema.

- Como también la potencial ventaja que tiene el uso de la aplicación dentro del entorno académico, sirviendo de herramienta para la enseñanza de la fotointerpretación.

5.2 Recomendaciones

- La visión de las imágenes en el sistema es dependiente del área de visión del monitor, mientras mayor tamaño tenga el monitor, mayor será el tamaño de las fotografías que se puedan desplegar. Puesto que el hecho de tratar con fotografías de gran tamaño sin perder la proporción en el sistema es bastante complejo, se deja a juicio del operador, el buscar fotografías que se ajusten al área de visión en el visor fotográfico y anaglifo.

- Fotografías con un paralaje relativo Pr bastante alto, no permiten una visualización correcta del anaglifo, para se ello se recomienda, usar fotografías con paralajes relativos de nivel medio o menor, para así tener una mejor visión estereoscópica.

BIBLIOGRAFIA

[SCHACH, 2005] STEPHEN R. SCHACH. "Análisis y Diseño Orientado a Objetos con UML y el Proceso Unificado**".** Primera Edición, McGraw-Hill Interamericana**.** México, 2005.

[LARMAN, 2003] LARMAN, CRAIG. "UML Y PATRONES. Una introducción al análisis y diseño orientado a objetos y al proceso unificado". Segunda Edición. PEARSON EDUCATION, S.A., Madrid, 2003.

[OMG, 2003] Object Management Group. "OMG Unified Modeling Language Specification". Versión 1.3, Marzo 2003. Documento en formato digital, revisado en http://www.omg.org/technology/documents/formal/uml.

[DAGOSTINI, 1984] Deagostini, D. 1984. "Introducción a la fotogrametría". C.I.A.F., Bogotá, 1984.

[WOLF, 1974] Wolf, P. R. 1974. Elements of photogrammetry. Mc Graw-Hill, New York.

[Internet 1] http://inicia.es/de/M_MS/vision3d.html

[Internet 2] http://www.tav.net/docs_imagen3d/index.htm

[Internet 3] http://personales.upv.es/rmullor/index.html

[Internet 4] http://www.trolltech.com

ANEXO A
CASOS DE USO

A continuación se muestran los casos de uso finales del Sistema de visualización de tridimensional de imágenes con uso de marca flotante:

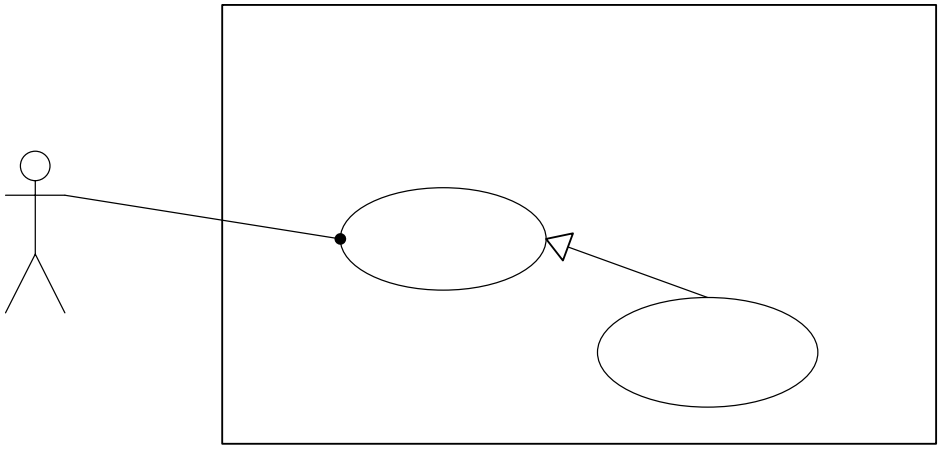

Figura Nº 41. Caso de Uso Crear Nuevo Proyecto

Actores:	Operador
Función:	El caso de uso *Crear Nuevo Proyecto* permite al Operador crear un nuevo proyecto que servirá de repositorio para las fotografías y los datos.
Descripción:	1. El operador selecciona las fotografías (ver caso de uso *Seleccionar Fotografías*). 2. El operador asigna un nombre al proyecto.

Tabla Nº 6. Descripción del Caso de Uso Crear Nuevo Proyecto

Actores:	Operador
Función:	El caso de uso *Seleccionar Fotografías* permite al Operador seleccionar las fotografías a utilizar.
Descripción:	1. Para la fotografía izquierda: 1.1. El sistema muestra una caja de dialogo para seleccionar el archivo de imagen. 1.2. El operador selecciona el archivo y presiona aceptar. 1.3. Si es un archivo valido el sistema muestra una vista previa de la fotografía izquierda. 2. Para la fotografía derecha: 2.1. El sistema muestra una caja de dialogo para seleccionar el archivo de imagen. 2.2. El operador selecciona el archivo y presiona aceptar. 2.3. Si es un archivo valido el sistema muestra una vista previa de la fotografía derecha.

Tabla Nº 7. Descripción del Caso de Uso Seleccionar Fotografías

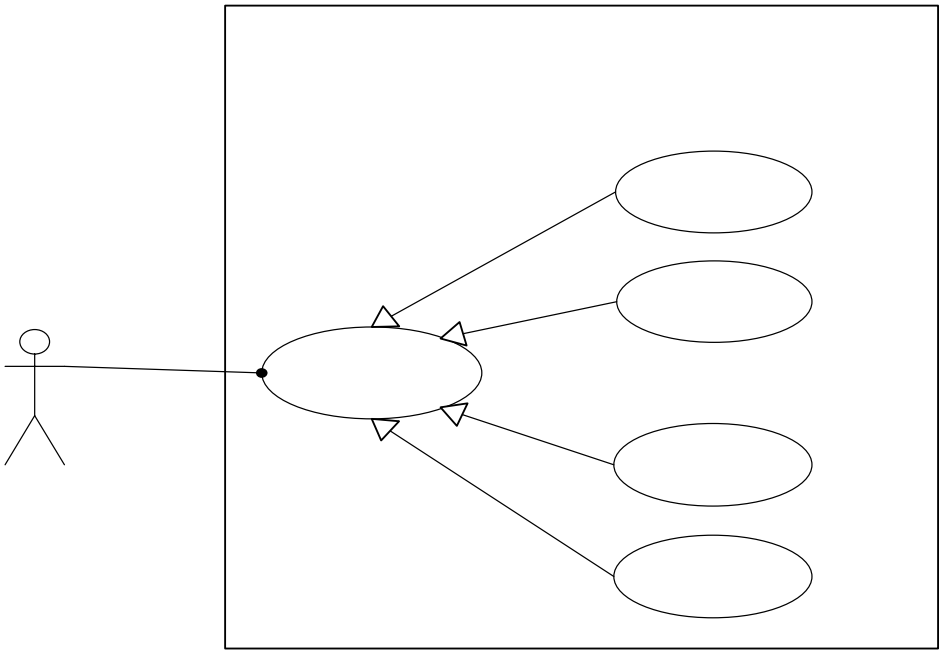

Figura Nº 42. Caso de Uso Configurar Parámetros

Actores:	Operador
Función:	El caso de uso *Configurar parámetros,* permite al Operador asignar los parámetros necesario para calcular las coordenadas del punto homologo.
Descripción:	1. La constante *mb,* se obtiene mediante el caso de uso configurar *mb.* 2. La constante *dpi,* se obtiene mediante el caso de uso configurar *dpi.* 3. La constante *Zp,* se obtiene mediante el caso de uso configurar *Zp.* 4. La constante *Pr,* se obtiene mediante el caso de uso configurar *Pr.*

Tabla Nº 8. Descripción del Caso de Uso Configurar Parámetros

Actores:	Operador
Función:	El caso de uso *Configurar mb,* permite al Operador asignar la constante de escala *mb.*
Descripción:	El Operador especifica el valor de la constante de escala *mb.*

Tabla Nº 9. Descripción del Caso de Uso Configurar mb

Actores:	Operador
Función:	El caso de uso *Configurar dpi,* permite al Operador asignar la constante de la fotografía *dpi.*
Descripción:	El Operador especifica el valor de la constante *dpi.*

Tabla Nº 10. Descripción del Caso de Uso Constante dpi

Actores:	Operador
Función:	El caso de uso *Configurar Zr,* permite al Operador asignar la constante *Zr.*
Descripción:	El Operador especifica el valor de la constante *Zr.*

Tabla Nº 11. Descripción del Caso de Uso Zr

Actores:	Operador
Función:	El caso de uso *Configurar Pr,* permite al Operador obtener una aproximación del parámetro *Pr.*

Descripción:	1. El sistema coloca ambas fotografías una al lado de la otra. 2. El sistema coloca sobre cada fotografía un sistema de coordenadas cuyo origen se encuentra en el centro de la fotografía respectiva. 3. El Operador sobre la fotografía derecha ubica el mismo punto central de la fotografía izquierda, arrastrando, sobre el eje horizontal (eje X) el eje vertical (eje Y), hacia la izquierda o derecha según sea el caso. 4. El sistema, una vez localizado el punto en la fotografía derecha. Se calcula la distancia entre los puntos solapando ambos sistemas de coordenadas.

Tabla Nº 12. Descripción del Caso de Uso Configurar Pr

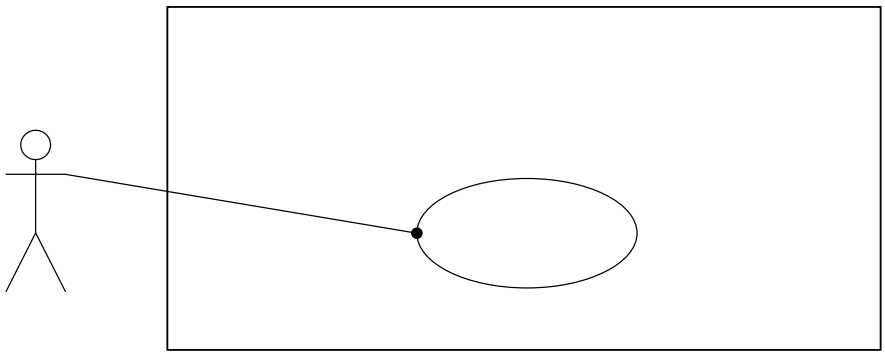

Figura Nº 43. Caso de Uso Fijar Sistema de Coordenadas

Actores:	Operador
Función:	El caso de uso *fijar sistema de coordenadas,* permite al Operador fijar el sistema de coordenadas que se usará como referencia para situar el punto homologo.

Descripción:	1. El Operador activa el modo de sistema de coordenadas sobre la fotografía izquierda. 2. El sistema visualiza el sistema de coordenadas sobre la fotografía izquierda. 3. El Operador haciendo click o doble click en un punto sobre la fotografía ajusta el origen del sistema de coordenadas.

Tabla Nº 13. Descripción del Caso de Uso Fijar Sistema de Coordenadas

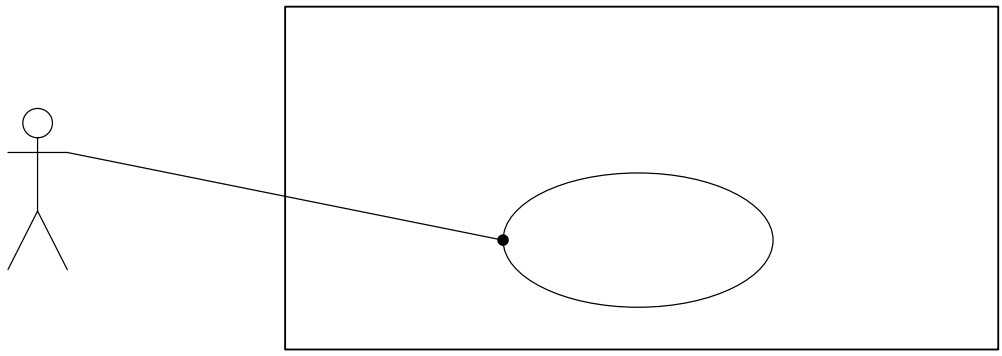

Figura Nº 44. Caso de Uso Fijar Punto Homologo en Modo Fotografía

Actor:	Operador
Función:	El caso de uso *Fijar Punto Homologo en Modo Fotografía,* permite al Operador fijar el punto homologo sobre la fotografía izquierda en modo fotografía.
Descripción:	1. El Operador activa el modo de sistema de coordenadas sobre la fotografía. 2. El sistema coloca 2 ejes referenciales (uno vertical y el otro horizontal) cuya intersección es el punto homologo, en el centro de la fotografía.

	3. El Operador haciendo click o doble click en un punto sobre la fotografía ajusta la posición del punto homologo.

Tabla Nº 14. Descripción del Caso de Uso Fijar Punto Homologo en Modo Fotografía

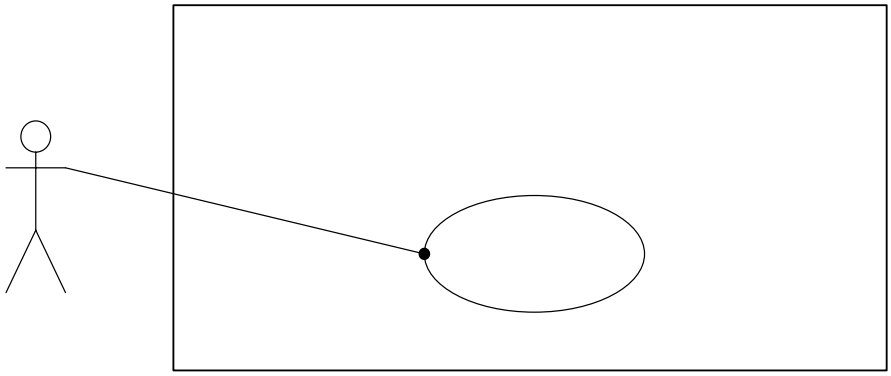

Figura Nº 45. Caso de Uso Visualizar Anaglifo

Actores:	Operador
Función:	El caso de uso *Visualizar Anaglifo,* permite al Operador visualizar la imagen anaglífa creada por el sistema a través de las fotografías aéreas del terreno.
Descripción:	4. El Operador selecciona modo Anaglifo. 5. El sistema muestra una caja de dialogo que contiene el visor anaglifo. 6. El sistema el sistema crea el anaglifo y lo muestra en el visor.

Tabla Nº 15. Descripción del Caso de Uso Visualizar Anaglifo

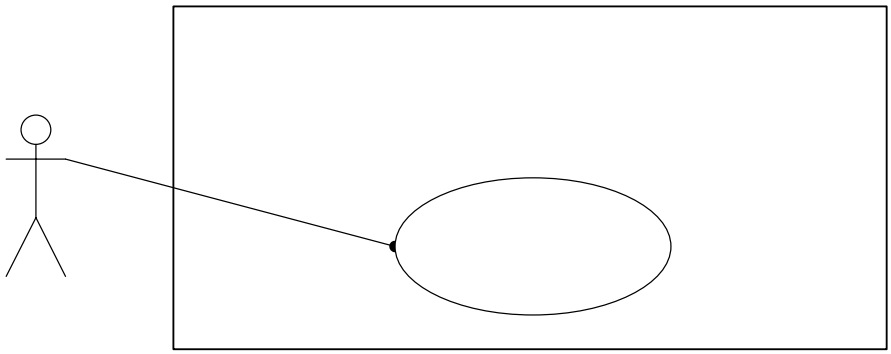

Figura Nº 46. Caso de Uso Asignar grado de solapamiento o separación horizontal

Actores:	Operador
Función:	El caso de uso *Asignar Grado de Solapamiento,* permite al Operador modificar el grado de solapamiento Z que tendrá la imagen anaglifa.
Descripción:	4. El Operador posiciona el cursor sobre la barra deslizadora o slider. 5. El Operador arrastra la barra deslizadora o mueve el scroll hacia arriba o abajo según sea el caso. 6. El sistema crea el anaglifo con el nuevo ajuste.

Tabla Nº 16. Descripción del Caso de Uso Asignar grado de solapamiento o separación horizontal

Operador

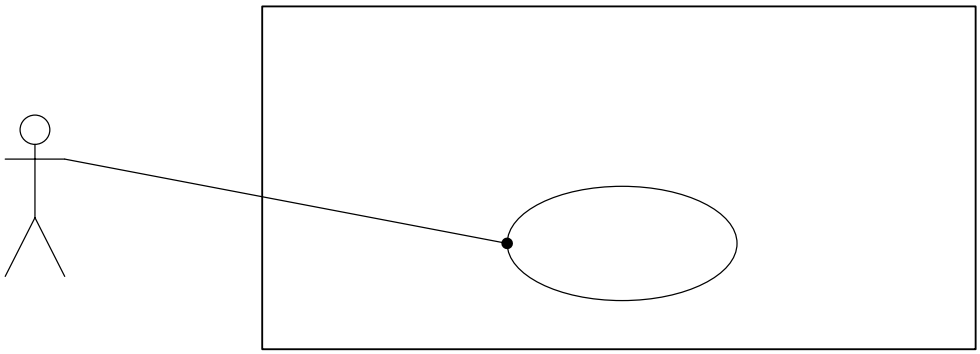

Figura Nº 47. Caso de Uso Recopilar Información de la Imagen Anaglifa

Actores:	Operador
Función:	El caso de uso *Recoger Información de la Imagen Anaglifo,* permite al Operador recoger puntos tridimensionales de la imagen anaglífa.
Descripción:	1. El Operador selecciona la opción Datos Recogidos. 2. El sistema muestra una caja de dialogo con la lista de puntos recopilados anteriormente. 3. El Operador presiona Agregar Nueva Coordenada. 4. El sistema agrega un nuevo Punto a la lista de datos recopilados.

Tabla Nº 18. Descripción del Caso de Uso Recopilar Información de la Imagen Anaglifa

Operador

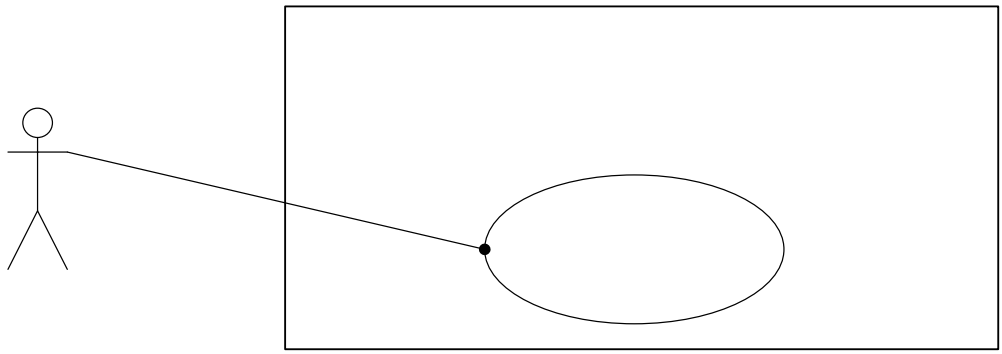

Figura Nº 48. Caso de Uso Fijar Punto Homologo en Modo Anaglifo

Actor:	Operador
Función:	El caso de uso *Fijar Punto Homologo en Modo Anaglifo,* permite al Operador fijar el punto homologo sobre la imagen anaglifa.
Descripción:	1. El Operador activa Mover punto homologo. 2. El sistema coloca 2 ejes referenciales (uno vertical y el otro horizontal) cuya intersección es el punto homologo. 3. El Operador da una nueva posición al punto, moviendo los ejes referenciales. 4. El Operador se selecciona fijar punto homologo.

Tabla Nº 19. Descripción del Caso de Uso Punto Homologo en Modo Anaglifo

Operador

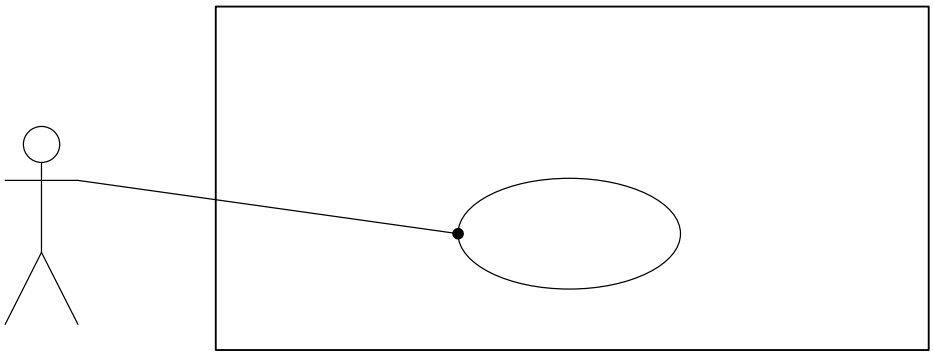

Figura Nº 49. Caso de Uso Guardar Proyecto

Actores:	Operador
Función:	El caso de uso *Guardar Proyecto,* permite al Operador almacenar las fotografías y los datos obtenidos de la imagen anaglífa.
Descripción:	1. El sistema crea un directorio con el nombre del proyecto y un archivo de extensión .pya 1.1. En el directorio almacena las fotografías. 1.2. En el archivo .pya almacena en binario los puntos obtenidos de la imagen anaglífa junto con los parámetros usados para obtener los datos.

Tabla Nº 20. Descripción del Caso de Uso Guardar Proyecto

ANEXO B

DIAGRAMAS DE COLABORACIÓN Y SECUENCIA

Se presentan los diagramas de colaboración y secuencia finales para cada uno de los casos de Uso presentados en el **Anexo A:**

Caso de Uso: Nuevo proyecto.

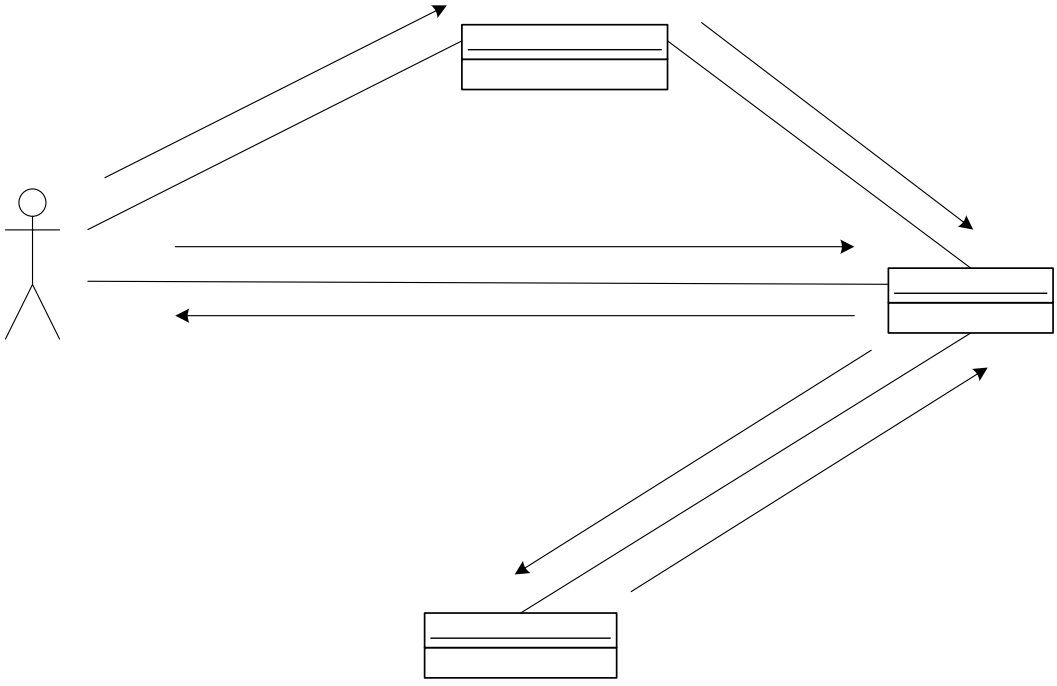

Figura Nº 50. Diagrama de colaboración, caso de uso nuevo proyecto.

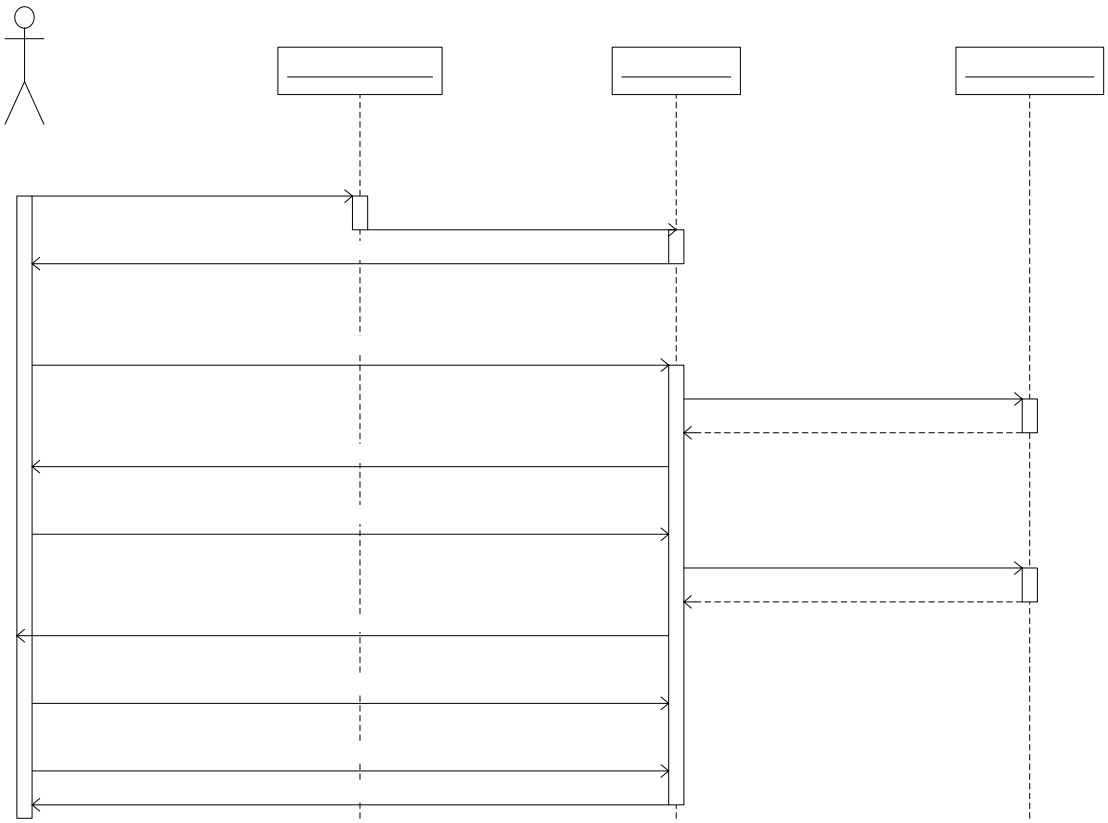

Figura Nº 51. Diagrama secuencia, caso de uso nuevo proyecto

Caso de uso: Ajustar Sistema de Coordenadas.

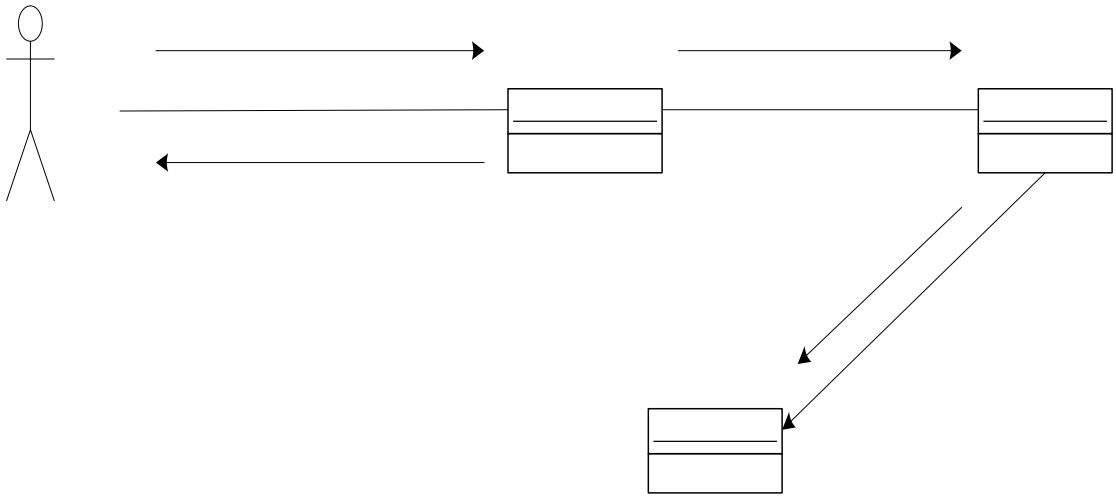

Figura Nº 52. Diagrama de colaboración, caso de uso ajustar el sistema de coordenadas.

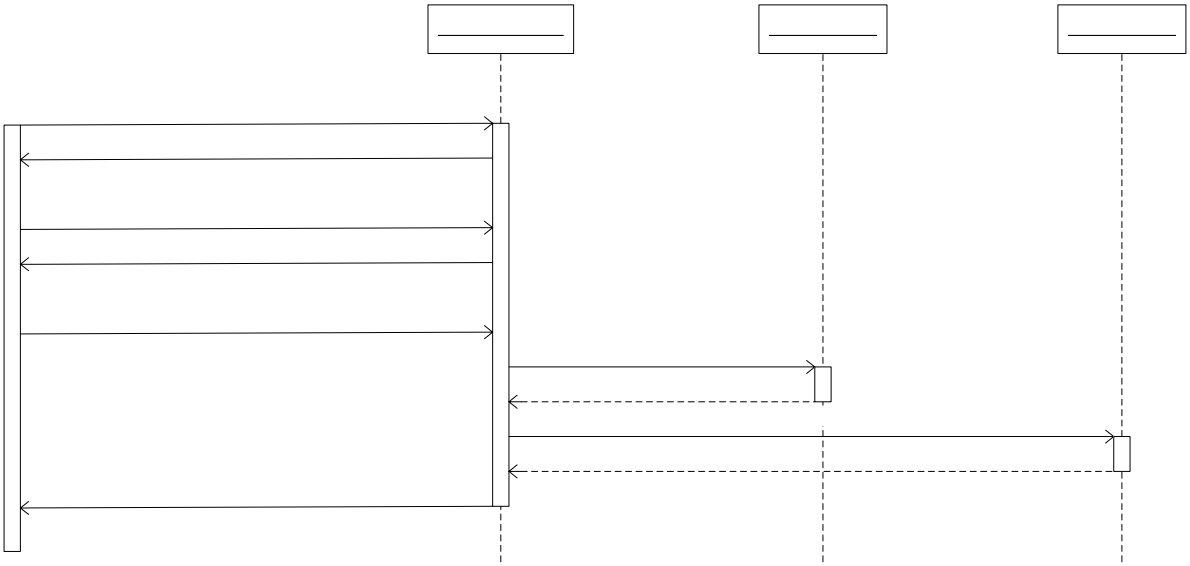

Figura Nº 53. Diagrama de secuencia caso de uso ajustar el sistema de coordenadas.

Caso de uso: Ajustar punto Homologo en modo Fotografía.

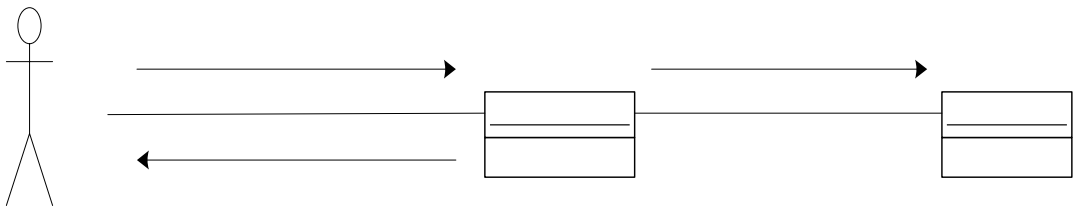

Figura Nº 54. Diagrama de colaboración caso de uso ajustar punto homologo en modo fotografía.

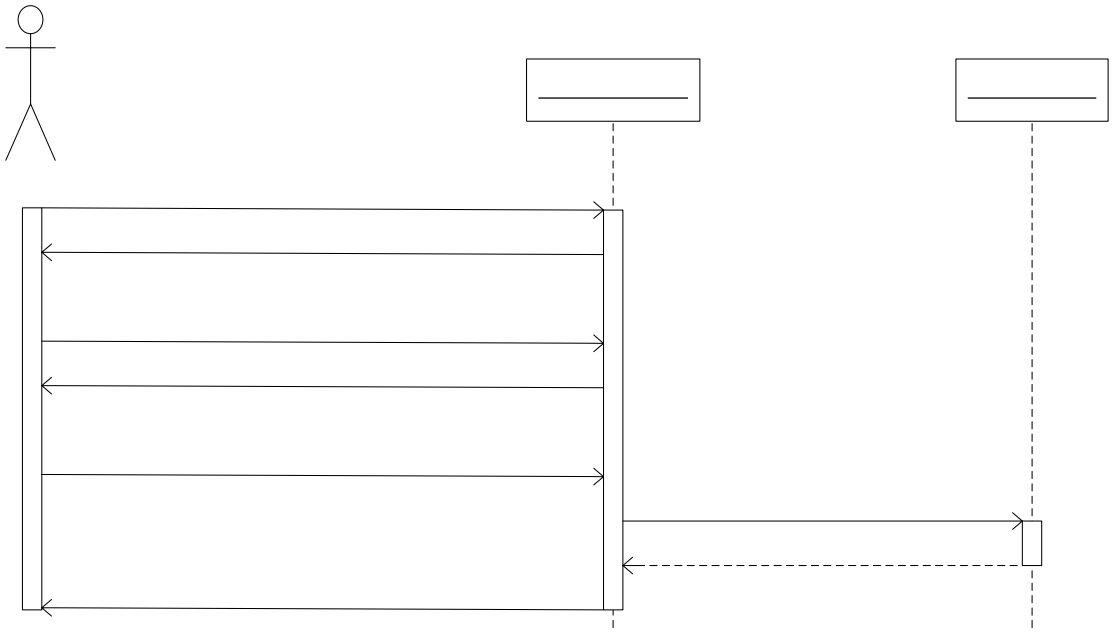

Figura Nº 55. Diagrama de secuencia caso de uso ajustar punto homologo en modo fotografía.

Caso de uso: Ajustar Punto Homologo en Modo Anaglifo.

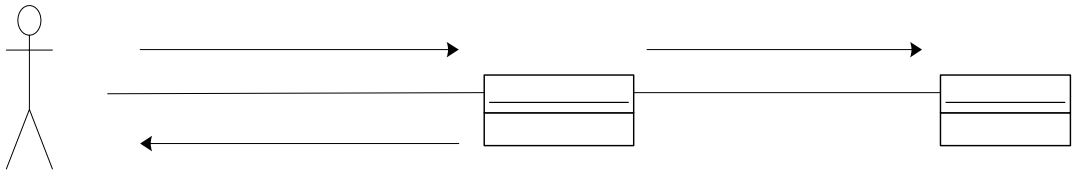

Figura Nº 56. Diagrama de colaboración caso de uso ajustar punto homologo en modo anaglifo.

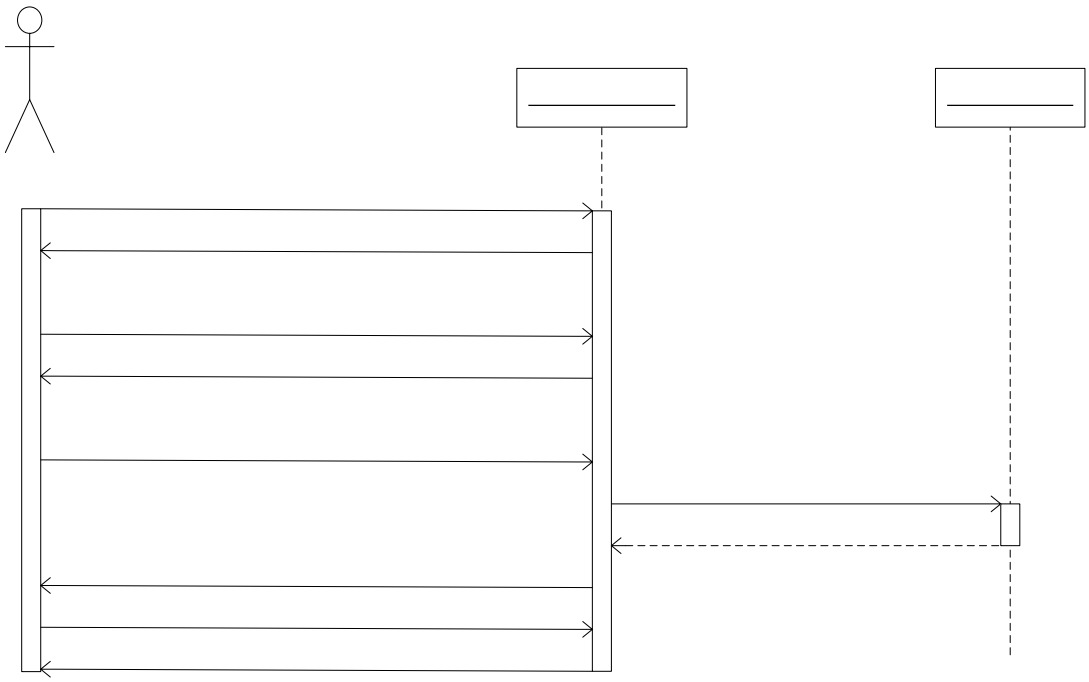

Figura Nº 57. Diagrama de secuencia caso de uso ajustar punto homologo en modo anaglifo.

Caso de uso: Visualizar Anaglifo.

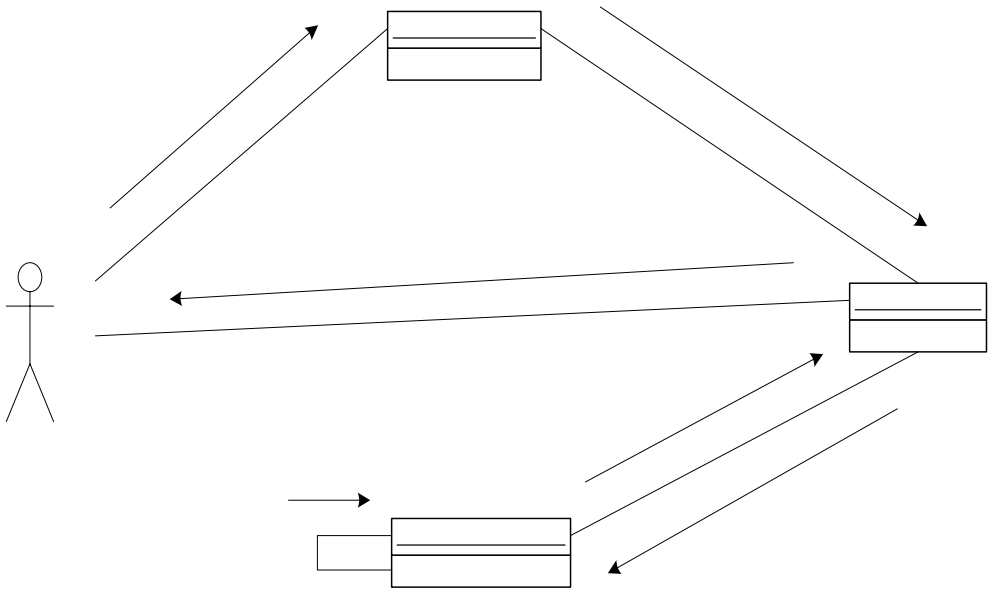

Figura Nº 58. Diagrama de colaboración caso de uso visualizar anaglifo.

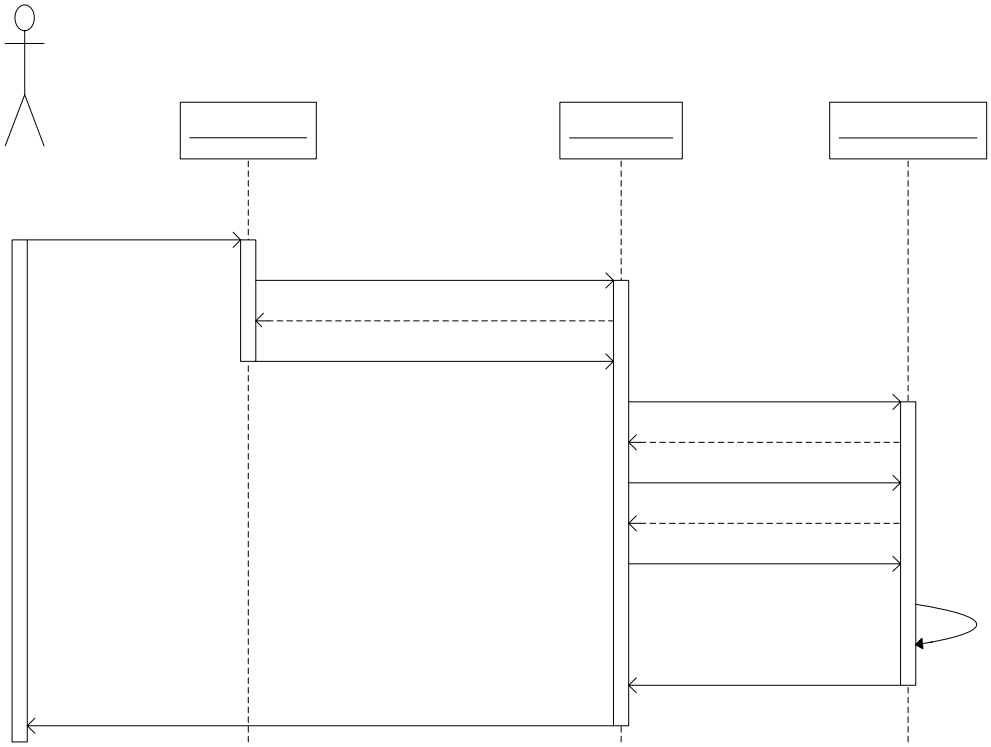

Figura Nº 59. Diagrama de secuencia caso de uso visualizar anaglifo.

Caso de uso: Guardar Proyecto.

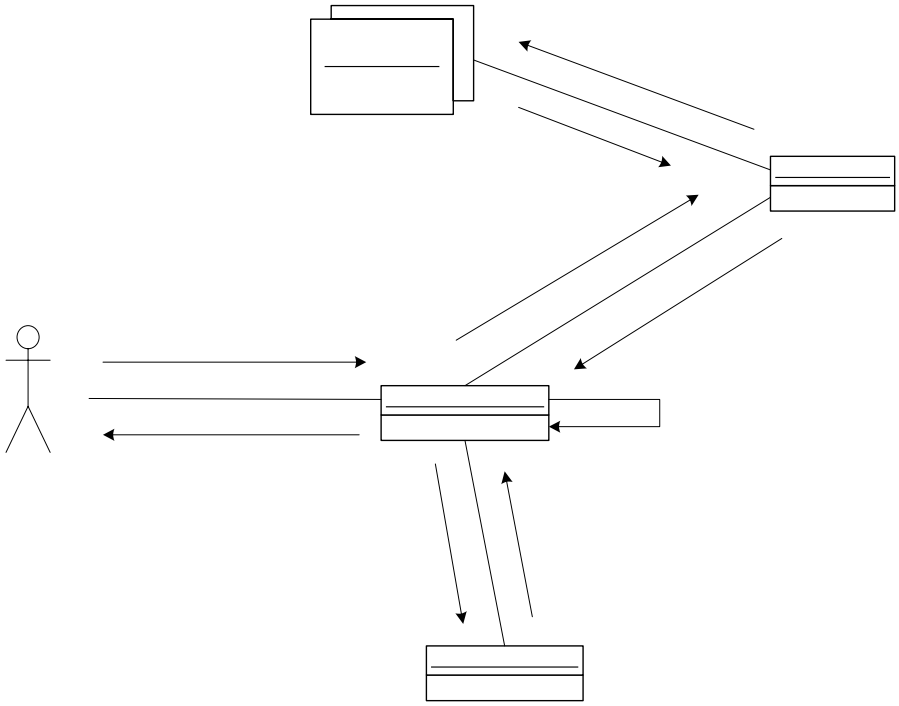

Figura Nº 60. Diagrama de colaboración caso de uso guardar proyecto.

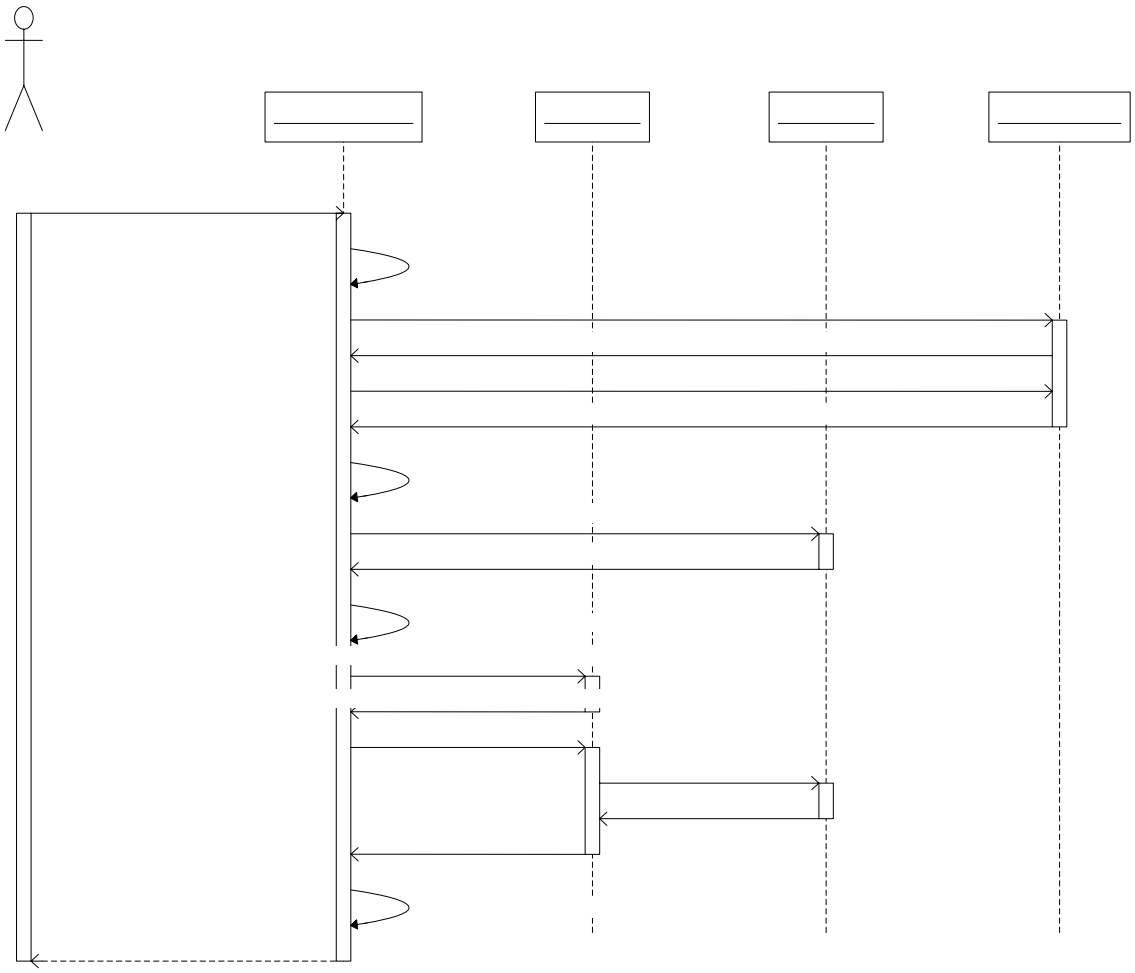

Figura Nº 61. Diagrama de secuencia caso de uso guardar proyecto.

Top Package::Operador

1: Selecciona

Caso de uso: Recopilar Información de la Imagen Anaglifa.

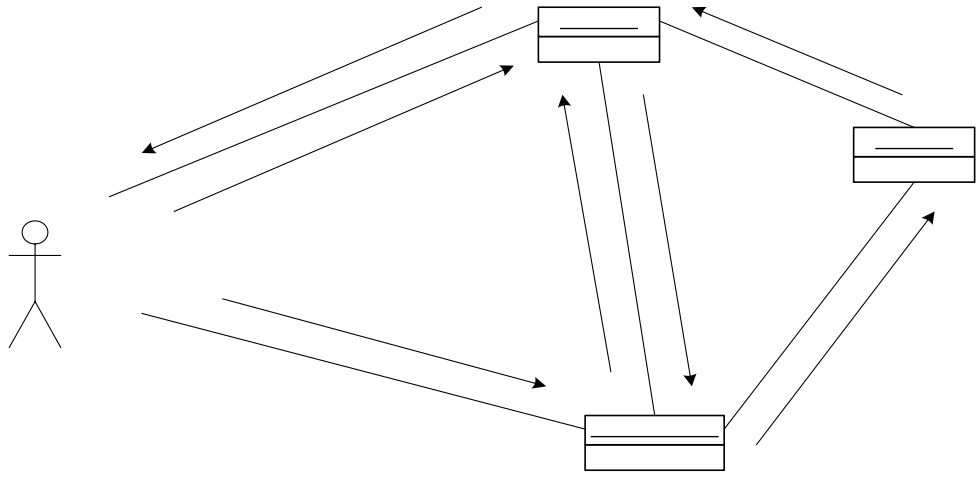

Figura Nº 62. Diagrama de colaboración caso de uso recopilar información de la imagen anaglifa.

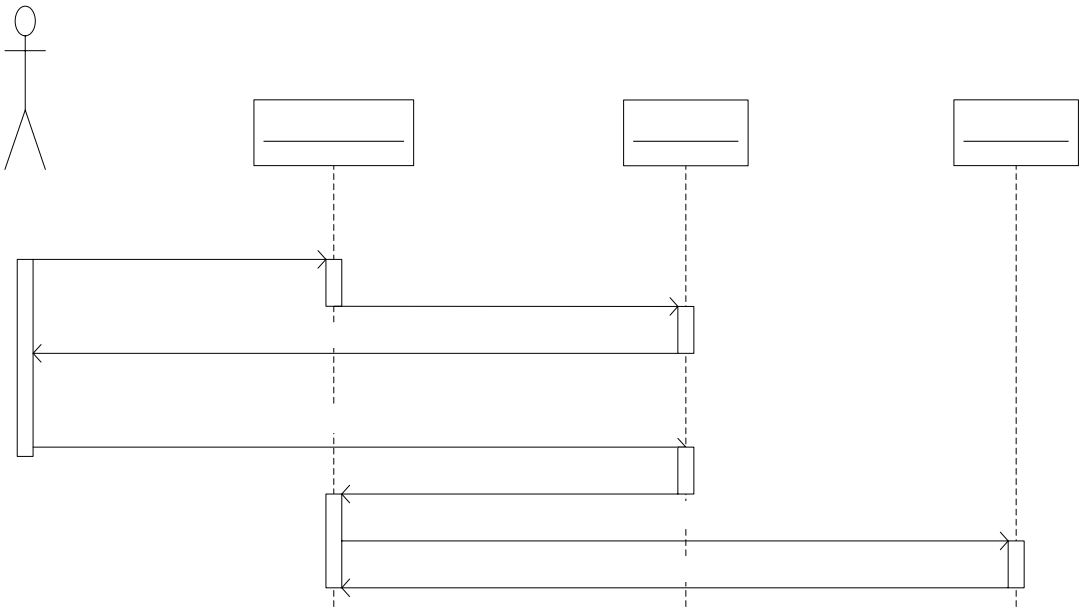

Figura Nº 63. Diagrama de secuencia caso de uso recopilar información de la imagen anaglifa.

Caso de uso: Asignara grado de Solapamiento.

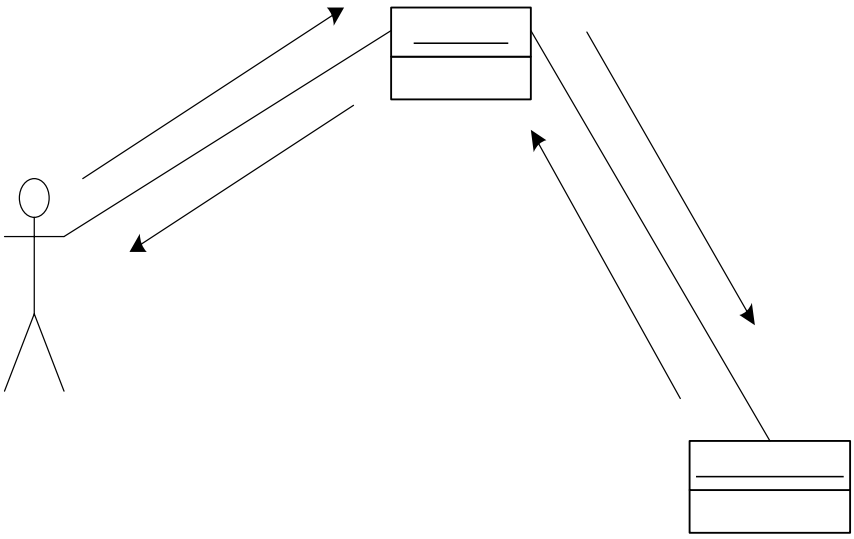

Figura Nº 64. Diagrama de colaboración caso de uso asignar grado de solapamiento.

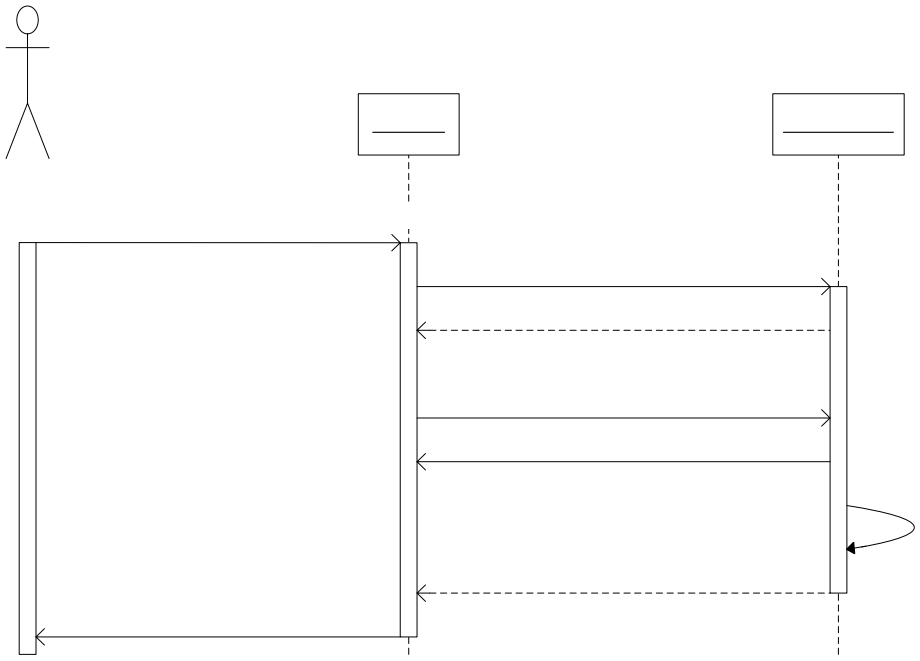

Figura Nº 65. Diagrama de secuencia caso de uso asignar grado de solapamiento.

ANEXO C
IMPLEMENTACIÓN DE LAS CLASES EN TDSO

#1 Clase QPoint3D : QPointF	
Estructura: Z : flotante de doble precisión (doble) **Especificación Sintáctica:** 1. QPoint3D() 2. QPoint3D(x : doble, y : doble, z : doble) 3. rz() 4. z() 5. setZ(z : doble) **Especificación Semántica:** int x=1, y=2, z=3; QPoint3D p3d_1, p3d_2(x,y,z); p3d_1.setZ(3) ;	- Z: Coordenada z del punto. - QPoint3D(): Constructor por defecto. - QPoint3D(x : doble, y : doble, z : doble): Constructor parametrico de la clase. - rz(): Retorna una referencia a la coordenada z del punto. - z(): Retorna la coordenada z del punto. - setZ(): Asigna la coordenada z al punto.

Tabla Nº 21. Especificación de la clase QPoint3D

#2 Clase QPRDialog : QDialog	
Estructura:	- cvLeft: Variable que sirve de visor

pr : entero i : entero cvLeft : Q3CVBase cvRight : Q3CVBase yAxisLeft : Q3CanvasLine xAxisLeft : Q3CanvasLine yAxisRight : Q3CanvasLine xAxisRight : Q3CanvasLine	para la fotografía izquierda. - cvRight: Variable que sirve de visor para la fotografía derecha. - yAxisLeft: Eje vertical de la fotografía izquierda. - xAxisLeft: Eje horizontal de la fotografía izquierda. - yAxisRight: Eje vertical de la fotografía derecha. - xAxisRight: Eje horizontal de la fotografía derecha. - pr: Paralaje relativo de las fotografías. - i: Mide el porcentaje de decremento en el ancho y largo de las fotografías, para ajustarlas al tamaño del visor sin afectar la proporción de aspecto de las mismas.
Especificación Sintáctica: 1. QPRDialog(parent = 0 : Puntero a QWidget, name = 0: Puntero a carácter, fl = 0: Qt::WFlags tipo enumerado) 2. openPhotoLeft(photoNameLeft : QString) 3. openPhotoRight(photoNameRight : QString) 4. returnPr() 5. systemLeft() 6. systemRight() 7. moveAxisX(e : entero) 8. aPushButtonSlot()	- QPRDialog(parent = 0 : Puntero a QWidget, name = 0: Puntero a carácter, fl = 0: Qt::WFlags tipo enumerado): Contructor parametrico y por defecto de la clase. - openPhotoLeft(photoNameLeft : QString): Método que carga la fotografía izquierda en el visor. - openPhotoRight(photoNameRight : QString): Método que carga la fotografía derecha en el visor. - returnPr(): Método que retorna el Valor Pr calculado de las

Especificación Semántica:
QWidget
QPRDialog *prDialog = new QPRDialog;
prDialog->show();
prDialog->openPhotoLeft(“foto1.png”);
prDialog->openPhotoLeft(“foto2.png”);
int pr = returnPr();	

	fotografías. - systemLeft(): Método que carga el eje vertical y horizontal en el visor izquierdo. - systemRight(): Método que carga el eje vertical y horizontal en el visor derecho. - moveAxisX(e : entero): Método que ajusta la posición del eje X y actualiza el valor de pr. - aPushButtonSlot(): Evento que se dispara luego de presionar "aPushButton". Calcula el Pr verdadero ajustándolo al tamaño real de las fotografías, para luego cerrar la ventana.

Tabla Nº 22. Especificación de la clase QPRDialog

#3 Clase QAMainWindow : QMainWindow	
Estructura:	- mBar: Puntero a objeto de tipo

mBar : Puntero a QMenuBar;
sBar: Puntero a QStatusBar;
menu : Puntero a QMenu;
dataCollectionAction : Puntero a QAction;
newProyectAction : Puntero a QAction;
openProyectAction : Puntero a QAction;
saveProyectAction : Puntero a QAction;
saveProyectAsAction : Puntero a QAction;
closeAction : Puntero a QAction;
doubleBufferAction : Puntero a QAction;
parametersAction : Puntero a QAction;
nameProyect : QString;
dDialog : Puntero a QDDialog;
pDialog : Puntero a QPDialog;
photoView : Puntero a QPhotoView;

Especificación Sintáctica:

1. QAMainWindow(parent = 0 : Puntero a QWidget, const char * name = 0 : Puntero a carácter, flags = 0 : Qt::WFlags)
2. newProyect()
3. openProyect()
4. saveProyect()
5. saveProyectAs()
6. data()
7. dobleBuffer(bool opcion)
8. parametersActivate()
9. createActions()
10. createMenus()

Especificación Semántica:

QAMainWindow window;

QMenuBar, añade la funcionalidad de barra de menú a la ventana.

- sBar: Puntero a objeto de tipo QStatusBar, añade la funcionalidad de la barra de estatus.

- menu: Puntero a objeto de tipo QMenu, añade un item a la barra de menú.

- dataCollectionAction: Puntero a objeto de tipo QAction que sirve de señal para lanzar el evento data().

- newProyectAction: Puntero a objeto de tipo QAction que sirve de señal para lanzar el evento newProyect().

- openProyectAction: Puntero a objeto de tipo QAction que sirve de señal para lanzar el evento openProyect().

- saveProyectAction: Puntero a objeto de tipo QAction que sirve de señal para lanzar el evento saveProyect().

- saveProyectAsAction: Puntero a objeto de tipo QAction que sirve de señal para lanzar el evento saveProyectAs().

- closeAction: Puntero a objeto de tipo QAction que sirve de señal para lanzar el evento close().

- doubleBufferAction: Puntero a objeto de tipo QAction que sirve de señal para lanzar el evento dobleBuffer(bool opcion).

- parametersAction : Puntero a objeto de tipo QAction que sirve de señal para lanzar el evento parametersActivate().

- nameProyect : Nombre del Proyecto.

	- dDialog : Ventana de tipo caja de dialogo que recolecta

Tabla Nº 23. Especificación de la clase QAMainWindow

#4 Clase QAnaglyphView : Q3CVBase	
Estructura: activeItem : Puntero a Q3CanvasItem **Especificación Sintáctica:** 1. Q3CVBase(parent=0 : puntero a QWidget, name=0 : puntero a carácter, flags = 0 : Qt::WFlags) 2. Q3CVBase(canvas : puntero a Q3Canvas, parent=0 : puntero a QWidget, name=0 : puntero a carácter, flags = 0 : Qt::WFlags) 3. getActiveItem() 4. setActiveItem(item : Q3CanvasItem) 5. showNewItem(item : Q3CanvasItem) 6. clear() **Especificación Semántica:** Q3CVBase *cvBase = new Q3CVBase(); Q3CanvasItem *cItem = new Q3CanvasItem(); cbBase->showNewItem(cItem);	- activeItem: Puntero a Q3CanvasItem, Referencia al último objeto añadido al visor, es decir, al objeto activo. - Q3CVBase(parent=0 : puntero a QWidget, name=0 : puntero a carácter, flags = 0 : Qt::WFlags): Constructor parametrico de la clase. - Q3CVBase(canvas : puntero a Q3Canvas, parent=0 : puntero a QWidget, name=0 : puntero a carácter, flags = 0 : Qt::WFlags): Constructor parametrico de la clase. - getActiveItem(): Retorna una referencia al ultimo objecto añadido al visor, es decir, al objeto activo. - setActiveItem(item : Q3CanvasItem): Asigna el objeto “item” como objeto activo. - showNewItem(item : Q3CanvasItem):): Asigna el objeto “item” como objeto activo y lo muestra en el visor. - clear(): Elimina todos los elementos que se encuentran en el visor.

Tabla Nº 24. Especificación de la clase QAnaglyphView

#5 Clase QPDialog : QDialog	

Estructura:
 	
pr : entero
zr : flotante de doble precisión
mb : entero
dpi : entero
pd : puntero a QPRDialog
nameProyect : QString
photoNameLeft : QString
photoNameRight : QString
 	
Especificación Sintáctica:
 	
1. QPDialog(parent = 0 : QWidget, name = 0 : puntero a carácter, modal = false : booleano, fl = 0 : Qt::WFlags)
2. init(_nameProyect : QString)
3. setPhotosNames(_photoNameLeft : QString, _photoNameRight : QString)
4. setParameters(_pr : entero, _zr : doble, _mb : entero, _dpi : entero)
5. getParameters(_pr : entero, _zr : doble, _mb : entero, _dpi : entero)
6. aPushButtonSlot()
7. prPushButtonSlot()
 	
Especificación Semántica:
 	
pDialog = new QPDialog;
pDialog->init(nameProyect);
pDialog->setPhotosNames(photoView->namePhotoLeft(), photoView->namePhotoRight());
	- pr: Atributo de tipo entero que almacena la constante Pr.
	
	- zr: Atributo de tipo doble que almacena la constante Zr.
	
	- mb: Atributo de tipo entero que almacena la constante Mb.
	
	- dpi: Atributo de tipo entero que almacena la constante dpi.
	
	- pd: Puntero a objeto QPRDialog. Un objeto QPRDialog es una caja de dialogo qu sirve para calcular Pr.
	
	- nameProyect: Cadena que almacena el nombre del proyecto.
	
	- photoNameLeft: Cadena que almacena el nombre de la fotografía izquierda.
	
	- photoNameRight: Cadena que almacena el nombre de la fotografía derecha.
	
	- QPDialog(parent = 0 : QWidget, name = 0 : puntero a carácter, modal = false : booleano, fl = 0 : Qt::WFlags): Constructor pareametrico de la clase.
	
	- init(_nameProyect : QString): Asigna el nombre del proyecto.
	
	- setPhotosNames(_photoNameLeft : QString, _photoNameRight : QString): Asigna los nombres a las fotografías izquierda y derecha.
	
	- setParameters(_pr : entero, _zr : doble, _mb : entero, _dpi : entero): Asigna valores a los atributos pr, zr, mb, dpi.

	- getParameters(_pr : entero, _zr : doble, _mb : entero, _dpi : entero): actualiza los parámetros con los valores de los atributos pr, zr, mb, dpi. - aPushButtonSlot(): Actualiza ciertos atributos y cierra la caja de dialogo. - prPushButtonSlot(): Muestra la caja de Dialogo pd.

Tabla Nº 25. Especificación de la clase QPDialog

#6 Clase QAWidget : QWidget	
Estructura: anaglyphView : Puntero a QAnaglyphView; slider : Puntero a QSlider; **Especificación Sintáctica:** 1. QAWidget(parent = 0 : Puntero a QWidget, f = 0 : Qt::WFlags) 2. init(photoNameLeft : QString, photoNameRight : QString, pos : QPoint, QPoint *zero, QDDialog *dDialog, int dpi, int mb, int pr, double zr) 3. returnPoint() 4. moveAnaglyph(int value) 5. keyPressEvent(QKeyEvent *e) 6. wheelEvent(QWheelEvent * e) **Especificación Semántica:** QAWidget *aw = new QAWidget; aw->init(nLeft, nRight, pHTranslate, pZTranslate, dDialog, dpi, mb, pr, zr);	- anaglyphView: Puntero a objeto QAnaglyphView, contiene la referencia al visor anaglifo. - QAWidget(parent = 0 : Puntero a QWidget, f = 0 : Qt::WFlags): Constructor parametrico de la clase. - init(QString photoNameLeft, QString photoNameRight, QPoint *pos, QPoint *zero, QDDialog *dDialog, int dpi, int mb, int pr, double zr): Inicializa todos los parámetros ncesarios para crear la imagen anaglifo, como también los parámetros para el calculo de la coordenada Z. - returnPoint(): Retorna el punto homologo actual. - moveAnaglyph(int value): Redibuja el anaglifo con el nuevo grado de solapamiento. - keyPressEvent(QKeyEvent *e):

	Controla el evento, presionar Esc, si se presiona se cierra la caja de dialogo. - wheelEvent(QWheelEvent * e): Controla el evento, mover scroll, si se mueve el scroll Redibuja el anaglifo con el nuevo grado de solapamiento.

Tabla Nº 26. Especificación de la clase QAWidget

#7 Clase QNPDialog : QDialog	

Estructura:
 	
cvLeft : Puntero a Q3CVBase
photoNameLeft : QString
cvRight : Puntero a Q3CVBase
photoNameRight : QString
nameProyect : QString
 	
Especificación Sintáctica:
 	
1. QNPDialog(parent = 0 : Puntero a QWidget, fl = 0 : Qt::WFlags)
2. getPhotoNameLeft()
3. getPhotoNameRight()
4. getNameProyect()
5. openPhotoLeft()
6. openPhotoRight()
7. aPushButtonSlot()
8. buscar()
 	
Especificación Semántica:
 	
QNPDialog *nw = new QNPDialog;
photoView->loadPhotoLeft(nw->getPhotoNameLeft());
photoView->loadPhotoRight(nw->getPhotoNameRight());
nameProyect= nw->getNameProyect();
	- cvLeft: Puntero a objeto Q3CVBase, contiene la referencia del visor de la fotografía izquierda.
	
	- photoNameLeft: Cadena que almacena el nombre de la fotografía izquierda.
	
	- cvRight: Puntero a objeto Q3CVBase, contiene la referencia del visor de la fotografía derecha.
	
	- photoNameRight: Cadena que almacena el nombre de la fotografía derecha.
	
	- QNPDialog(parent = 0 : Puntero a QWidget, fl = 0 : Qt::WFlags): Constructor parametrico de la clase.
	
	- getPhotoNameLeft(): Retorna el nombre de la fotografía izquierda.
	
	- getPhotoNameRight(): Retorna el nombre de la fotografía derecha.
	
	- getNameProyect(): Retorna el nombre del proyecto.

	- openPhotoLeft(): Obtiene el nombre de la fotografía izquierda. - openPhotoLeft(): Obtiene el nombre de la fotografía derecha. - aPushButtonSlot(): Valida que esten las 2 fotografías y cierra la caja de dialogo. - buscar(): Selecciona el directorio donde se va a guardar el proyecto.

Tabla Nº 27. Especificación de la clase QNPDialog

#8 Clase QPhotography	
Estructura: dataFileName : QString; pixmap : Puntero a Q3CanvasPixmap; **Especificación Sintáctica:** 1. QPhotography() 2. QPhotography(datafilename : QString, format = 0 : Puntero a carácter) 3. getDataFileName() 4. getPixmap() 5. saveAs(datafilename : QString, format : Puntero a carácter, quality = -1 : entero) 6. loadPhoto(datafilename : QString) 7. isNull() 8. width() 9. height() **Especificación Semántica:** QPhotography photoLeft; QPhotography photoRight; photoLeft.loadPhoto(n); QString name =	- dataFileName: Cadena que almacena nombre del archivo de imagen. - Pixmap: Puntero a Q3CanvasPixmap, contiene la referencia al archivo de imagen. - QPhotography(): Constructor por defecto de la clase. - QPhotography(datafilename : QString, format = 0 : Puntero a carácter): Constructor parametrico de la clase. - getDataFileName(): Retorna el nombre del archivo de imagen. - getPixmap(): Retorna el archivo de imagen como objeto Q3CanvasPixmap; - saveAs(datafilename : QString, format : Puntero a carácter, quality = -1 : entero): Guarda la imagen en disco. - loadPhoto(datafilename : QString): Carga el archivo de imagen.

photoLeft.getDataFileName();	- isNull(): Retorna cierto si el archivo de imagen no es valido. - width(): Retorna el ancho de la fotografía. - height(): Retorna el largo de la fotografía.

Tabla Nº 28. Especificación de la clase QPhotography

IMPLEMENTACION DE LOS METODOS DE LAS CLASES EN TDSO

newProyect() **{ Método que muestra una ventana para el ingreso de un nuevo proyecto}**
Pre-condición Post-condición:
QNPDialog *nw = new QNPDialog Si(nw->exec() = QDialog::Accepted) Entonces 2.1 delete dDialog; 2.2 dDialog = new QDDialog; 2.3 dDialog->LLPushButtonSlot(); 2.4 photoView->loadPhotoLeft(nw->getPhotoNameLeft()); 2.5 photoView->loadPhotoRight(nw->getPhotoNameRight()); 2.6 nameProyect= nw->getNameProyect(); 2.7 photoView->setNameProyect(nameProyect) 2.8 entero pr = 0, mb = 1, dpi = 1; 2.9 flotante de doble presicion zr = 0; 2.10 photoView->setParameters(pr,zr,mb,dpi); 3. Liberar nw;

Tabla Nº 29. Método newProyect de la clase QAMainWindow

data() **{ Método que muestra la ventana que contiene los datos registrados }**
Pre-condición: Post-condición:
1. dDialog->exec() 2. fin_metodo

Tabla Nº 30. Método data de la clase QAMainWindow

dobleBuffer(booleano opcion) **{ Método que activa la opción de un buffer doble para la generación del anaglifo en tiempo real }**
Pre-condición: Post-condición:

```
photoView->setDoubleBuffer(opcion);
fin_metodo
```

Tabla Nº 31. Método dobleBuffer de la clase QAMainWindow

parametersActivate() **{ Método que muestra una ventana para la activación de los parámetros}**
Pre-condición: Post-condición:

```
Si ( nameProyect = " " ) entonces
      QMessageBox::information( this, "anaglyph", "Se debe abrir un proyecto con
           anterioridad" );
      Retornar
fin_si
pDialog = new QPDialog;
pDialog->init(nameProyect);
pDialog->setPhotosNames(photoView->namePhotoLeft(),photoView->namePhotoRight());
entero pr, mb, dpi;
flotante de doble presicion zr;
photoView->getParameters(pr,zr,mb,dpi);
pDialog->setParameters(pr,zr,mb,dpi);
Si ( pDialog->exec()==QDialog::Accepted ) entonces
      pDialog->getParameters(pr,zr,mb,dpi);
      photoView->setParameters(pr,zr,mb,dpi);
liberar pDialog;
fin_metodo
```

Tabla Nº 32. Método parametersActivate de la clase QAMainWindow

createActions() **{ Método que maneja el evento de crear una accion }**
Pre-condición: Post-condición:

```
1. newProyectAction = new QAction( tr("&Nuevo"), this );
2. newProyectAction->setShortcut(QKeySequence(tr("Ctrl+N")));
3. newProyectAction->setStatusTip(tr("Abrir un nuevo proyecto"));
4. connect( newProyectAction, SIGNAL(triggered()), this, SLOT(newProyect()) );
5. openProyectAction = new QAction( tr("&Abrir"), this );
6. openProyectAction->setShortcut(QKeySequence(tr("Ctrl+A")));
7. openProyectAction->setStatusTip(tr("Abrir un proyecto existente"));
```

```
8.  connect( openProyectAction, SIGNAL(triggered()), this, SLOT(openProyect()) );
9.  saveProyectAction = new QAction( tr("&Guardar"), this );
10. saveProyectAction->setShortcut(QKeySequence(tr("Ctrl+G")))
11. saveProyectAction->setStatusTip(tr("Guardar un proyecto actual"));
12. connect( saveProyectAction, SIGNAL(triggered()), this, SLOT(saveProyect()) );
13. saveProyectAsAction = new QAction( tr("Guardar &como..."), this );
14. saveProyectAsAction->setShortcut(QKeySequence(tr("Ctrl+C")));
15. saveProyectAsAction->setStatusTip(tr("Guardar un proyecto actual"));
16. connect( saveProyectAsAction, SIGNAL(triggered()), this, SLOT(saveProyectAs()) );
17. closeAction = new QAction(tr("&Salir"), this);
18. closeAction->setShortcut(tr("Ctrl+S"));
19. closeAction->setStatusTip(tr("Salir de la Aplicación"));
20. connect( closeAction, SIGNAL(triggered()), this, SLOT(close()) );
21. doubleBufferAction = new QAction(tr("Doble &Buffer"), this);
22. doubleBufferAction->setCheckable(true);
23. doubleBufferAction->setShortcut(QKeySequence(tr("Ctrl+B")));
24. doubleBufferAction->setStatusTip(tr("Activa/Desactiva Double Buffering"));
25. connect( doubleBufferAction, SIGNAL(triggered(bool)), this, SLOT(dobleBuffer(bool))
    );
26. dataCollectionAction = new QAction( tr("&Datos Recogidos"), this );
27. dataCollectionAction->setShortcut(QKeySequence(tr("Ctrl+D")));
28. dataCollectionAction->setStatusTip(tr("Descripcion de la información recogido y sus
    respectivos parametros"));
29. connect( dataCollectionAction, SIGNAL(triggered()), this, SLOT(data()) );
30. parametersAction = new QAction( tr("Asignar &Parametros"), this );
31. parametersAction->setShortcut(QKeySequence(tr("Ctrl+P")));
32. parametersAction->setStatusTip( tr("Datos Recogidos de las Fotografias") );
33. connect( parametersAction, SIGNAL(triggered()), this, SLOT(parametersActivate()) );
34. fin_metodo
```

Tabla Nº 33. Método createActions de la clase QAMainWindow

setNameLeft(cadena _name) **{ Método que asigna el nombre de la fotografa izquierda}**
Pre-condición: Post-condición:
1. photoNameLeft=_name; 2. fin_metodo

Tabla Nº 34. Método setNameLeft de la QAnaglyphView

setNameRight(cadena _name) **{ Método que asigna el nombre de la fotografa derecha}**
Pre-condición: Post-condición:
1. photoNameRight=_name 2. fin_metodo

Tabla Nº 34. Método setNameRight de la clase QAnaglyphView

setZ(entero _z) **{ Método que asigna la coordenada z}**
Pre-condición: Post-condición:
1. z=_z; 2. fin_metodo

Tabla Nº 35. Método setZ de la clase QAnaglyphView

Entero getZ() **{ Método que asigna la coordenada z}**
Pre-condición: Post-condición:
1. return z 2. fin_metodo

Tabla Nº 36. Método getZ de la clase QAnaglyphView

Booleano setParameters(entero &_pr, flotante doble &_zr, entero &_mb, entero &_dpi) **{ Método que asigna los parámetros del sistema}**
Pre-condición: Post-condición:
1. pr = _pr; 2. zr = _zr; 3. mb = _mb; 4. dpi = _dpi; 5. Retornar !((pr = 0) ó (zr = 0) ó (mb = 0) ó (dpi = 0)); 6. fin_metodo

Tabla Nº 37. Método setParameters de la clase QAnaglyphView

Flotante doble presicion deltaHra() { { Método que asigna los parámetros del sistema}
Pre-condición: Post-condición:
1. flotante doble presiion dHra = 0 2. flotante doble presicion ZR = ((100*zr)/INCHTOCM)*dpi; 3. Si (z > 0) entonces 3.1 dHra = ((double)ZR*(z-pr))/((double)z); 4. retornar (dHra/(double)dpi)*INCHTOCM*((double)mb); 5. fin_metodo

Tabla Nº 38. Método deltaHra de la clase QAnaglyphView

setDDialog(QDDialog *&_dDialog) { Método que asigna la ventana de datos recogidos}
Pre-condición: Post-condición:
1. dDialog = _dDialog; 2. disconnect(_dDialog, SIGNAL(addToList()), this, SLOT(addPoint())); 3. connect(_dDialog, SIGNAL(addToList()), this, SLOT(addPoint())); 4.

Tabla Nº 39. Método setDialog de la clase QAnaglyphView

Puntero al objeto QPoint returnPoint() { Método que devuelve un punto}
Pre-condición: Post-condición:
1. return (new QPoint(pos->x()-(width()-image.width())/2, pos->y()-(height()-image.height())/2)); 2. fin_metodo

Tabla Nº 40. Método returnPoint de la clase QAnaglyphView

init(puntero al objeto QPoint p, puntero al objeto QPoint zo, puntero al objeto QSlider slider)

{ Método que inicializa las variables }

Pre-condición:
Post-condición:

```
1.  image.load(photoNameLeft);
2.  aux_right.load(photoNameRight);
3.  aux = image;
4.  createAnaglyph(image, aux, aux_right, z);
5.  setCanvas(new Q3Canvas(width()-5,height()-5));
6.  canvas()->setDoubleBuffering(verdadero);
7.  canvas()->setBackgroundColor(Qt::black);
8.  cpa = new Q3CanvasPixmapArray();
9.  cpa->setImage(0,new Q3CanvasPixmap(image));
10. showNewItem( ( cs = new Q3CanvasSprite(cpa,canvas()) ) );
11. activeItem->move( (width()-image.width())/2, (height()-image.height())/2 );
12. activeItem->setZ(50);
13. point = new Q3CanvasEllipse( 7, 7, canvas() );
14. point->setBrush(QBrush(Qt::white, Qt::SolidPattern));
15. showNewItem(point);
16. activeItem->move( p->x()+(width()-image.width())/2, p->y()+(height()-image.height())/2
        );
17. activeItem->setZ(100);
18. pos = p;
19. zero = zo;
20. pos->setX( p->x()+(width()-image.width())/2 );
21. pos->setY( p->y()+(height()-image.height())/2 );
22. Q3CanvasEllipse *pa = new Q3CanvasEllipse( 7, 7, canvas() );
23. pa->setBrush(QBrush(Qt::blue, Qt::SolidPattern));
24. showNewItem(pa);
25. activeItem->move( zo->x()+(width()-image.width())/2, zo->y()+(height()-
        image.height())/2 );
26. activeItem->setZ(100);
27. slider->setMinimum(-image.width());
28. slider->setMaximum(image.width());
29. slider->setTickmarks(QSlider::Both);
30. slider->setTickInterval(10);
31. origen=worldMatrix();
32. origen=origen.translate( (-1)*(zo->x()+(width()-image.width())/2), (-1)*(zo-
        >y()+(height()-image.height())/2) );
33. fin_metodo
```

Tabla Nº 41. Método init de la clase QAnaglyphView

createAnaglyph(QImage &image, QImage aux, QImage aux_right, entero z)
{ Método que permite crear un anaglifo}

Pre-condición:
Post-condición:

```
1.  QColor color, color_right;
2.  uint *ptr = nulo, *ptr_right = nulo;
3.  repita_para (entero i = 0;  i < image.width();  i++)
     3.1. repita_para (entero j = 0; j < image.height();  j++)
          3.1.1     ptr = reinterpret_cast<uint *>(aux.scanLine(j)) + i;
          3.1.2     color.setRgba(*ptr);
          3.1.3     si ( color.alpha() ) entonces
               3.1.3.1  Si( i>=z ) entonces
                    3.1.3.1.1      ptr_right = reinterpret_cast<uint *>(aux_right.scanLine(j)) +
                                   (i-z);
                    3.1.3.1.2      color_right.setRgba(*ptr_right);
                    3.1.3.1.3      *ptr = qRgba(color.red(),color_right.green(),0,color.alpha());
                    3.1.3.1.4      Sino entonces
                         3.1.3.1.4.1   *ptr = qRgba(color.red(),0,0,color.alpha());
                    3.1.3.1.5      fin_sino
               3.1.3.2  fin_si
          3.1.4     fin_si
     3.2    fin_repita_para
4.  fin_repita_para
5.  fin_metodo
```

Tabla Nº 42. Metodo createAnaglyph de la clase QAnaglyphView

stayPointActivate()
{ Método que permite activar la vision del punto homologo}

Pre-condición:
Post-condición:

```
1.  point->move( pos->x(), pos->y() );
2.  Si ( ( hLineReference ! = nulo )  y ( vLineReference ! = nulo ) )
      2.1  hLineReference->hide();
      2.2  vLineReference->hide();

3.  fin_si
4.  point->show();
5.  canvas()->update();
6.  pfloat=false;
7.  movePointAction->setEnabled(true);
8.  stayPointAction->setEnabled(false);
9.   fin_metodo
```

Tabla Nº 43. Método stayPointActivate de la clase QAnaglyphView

movePointActivate() **{ Método que permite mover el punto homologo}**
Pre-condición: Post-condición:
1. Si((hLineReference = nulo) y (vLineReference = nulo)) entonces 1.1 vLineReference = new Q3CanvasLine(canvas()); 1.2 Q_CHECK_PTR(vLineReference); 1.3 vLineReference->setPoints(pos->x(), 0, pos->x(), height()); 1.4 vLineReference->setZ(150); 1.5 showNewItem(vLineReference); 1.6 hLineReference = new Q3CanvasLine(canvas()); 1.7 Q_CHECK_PTR(hLineReference); 1.8 hLineReference->setPoints(0, pos->y(), width(), pos->y()); 1.9 hLineReference->setZ(150); 1.10 showNewItem(hLineReference); 2. fin_si 3. point->hide(); 4. hLineReference->show(); 5. vLineReference->show(); 6. canvas()->update(); 7. pfloat=rae; 8. stayPointAction->setEnabled(verdadero); 9. movePointAction->setEnabled(falso); 10. fin_metodo

Tabla Nº 44. Método movePointActivate de la clase QAnaglyphView

dDialogActivate() **{ Método que muestra una caja de dialogo que permite la obtención de coordenadas}**
Pre-condición: Post-condición:
1. dDialog->exec() 2. fin_metodo

Tabla Nº 45. Método dDialogActivate de la clase QAnaglyphView

mouseDoubleClickEvent(puntero a objeto QmouseEvent e) **{ Método que muestra una caja de dialogo que permite la obtención de coordenadas}**

Pre-condición: Post-condición:
1. Si ((hLineReference = nulo) ó (vLineReference = nulo) ó (e->button() = Qt::RightButton)) entonces 1.1 retornar 2. Qpoint aux; 3. Qpoint3D pmts; 4. si (pfloat) entonces 4.1 hLineReference->setPoints(0,e->y(),width(),e->y()); 4.2 vLineReference->setPoints(e->x(),0,e->x(),height()); 4.3 pos->setX(e->x()); 4.4 pos->setY(e->y()); 4.5 aux=origen.map(*pos); 4.6 aux.setY(-aux.y()); 4.7 pmts.setX(((((double) aux.x())/dpi)*INCHTOCM*((double)mb))/100); 4.8 pmts.setY(((((double) aux.y())/dpi)*INCHTOCM*((double)mb))/100); 4.9 pmts.setZ(deltaHra()); 4.10 QmessageBox::information(this,"","Punto ("+Qstring::number(pmts.x())+" mts; "+Qstring::number(pmts.y())+" mts; "+Qstring::number(pmts.z())+" mts)" 6. fin_si 7. canvas()->update(); 8. fin:_metodo

Tabla Nº 46. Método mouseDoubleClickEvent de la clase QAnaglyphView

mousePressEvent (puntero a objeto QMouseEvent * e) **{ Método que maneja el evento presionar el boton izquierdo }**
Pre-condición: Post-condición:
1. Si ((hLineReference==NULL) || (vLineReference==NULL) || (e->button()==Qt::RightButton)) entonces 1.1 retornar 2. fin_si 3. QPoint aux; 4. Si (pfloat) entonces 4.1 hLineReference->setPoints(0,e->y(),width(),e->y()); 4.2 vLineReference->setPoints(e->x(),0,e->x(),height()); 4.3 pos->setX(e->x()); 4.4 pos->setY(e->y()); 4.5 aux=origen.map(*pos); 4.6 aux.setY(-aux.y());

```
        4.7  pmts.setX( ( ( ( (double) aux.x() )/dpi )*INCHTOCM*( (double)mb  ) )/100 );
        4.8  pmts.setY( ( ( ( (double) aux.y() )/dpi )*INCHTOCM*( (double)mb  ) )/100 );
        4.9  pmts.setZ( deltaHra() );
        4.10     setToolTip("Punto ( "+QString::number( pmts.x() )+" mts; "+QString::number(
             pmts.y() )+" mts; "+QString::number( pmts.z() )+" mts )");
5.   fin_si
6.   canvas()->update();
7.   fin_metodo
```

Tabla Nº 47. Método mousePressEvent de la clase QAnaglyphView

contentsContextMenuEvent(puntero a objeto QContextMenuEvent *event) **{ Método que meneja el evento de visualizar el menú de contexto}**
Pre-condición: Post-condición:
1. contextMenu->popup(event->pos()); 2. contextMenu->show(); 3. fin_metodo

Tabla Nº 48. Método contentsContextMenuEvent de la clase QAnaglyphView

addPoint() ( ) **{ Método que permite añadir puntos }**
Pre-condición: Post-condición:
1. QPoint aux = origen.map(*pos); 2. aux.setY(-aux.y()); 3. QPoint3D p3d; 4. p3d.setX(((((double) aux.x())/dpi)*INCHTOCM*((double)mb))/100); 5. p3d.setY(((((double) aux.y())/dpi)*INCHTOCM*((double)mb))/100); 6. p3d.setZ(deltaHra()); 7. dDialog->addPoint(p3d); 8. fin_metodo

Tabla Nº 49. Método addPoint de la clase QAnaglyphView

createActions() **{ Método que crear una nueva accion }**

Pre-condición: Post-condición:
1. stayPointAction = new QAction(tr("&Fijar Punto Homologo"), this); 2. stayPointAction->setShortcut(tr("Ctrl+F")); 3. stayPointAction->setStatusTip(tr("Activa/Desactiva Punto Homologo")); 4. connect(stayPointAction, SIGNAL(triggered()), this, SLOT(stayPointActivate())); 5. movePointAction = new QAction(tr("&Mover punto homologo"), this); 6. movePointAction->setShortcut(tr("Ctrl+M")); 7. movePointAction->setStatusTip(tr("Activa/Desactiva Ejes de Punto Homologo")); 8. connect(movePointAction, SIGNAL(triggered()), this, SLOT(movePointActivate())); 9. dataCollectionAction = new QAction(tr("Datos &Recogidos"), this); 10. dataCollectionAction->setShortcut(tr("Ctrl+R")); 11. dataCollectionAction->setStatusTip(tr("Muestra todos los datos recogidos a través del anaglifo")); 12. connect(dataCollectionAction, SIGNAL(triggered()), this, SLOT(dDialogActivate())); 13. fin_metodo

Tabla Nº 50. Método createActions de la clase QAnaglyphView

createMenus() **{ Método que despliega una ventana para crear un nuevo menú }**
Pre-condición: Post-condición:
1. contextMenu->addAction(stayPointAction); 2. contextMenu->addAction(movePointAction); 3. contextMenu->addAction(dataCollectionAction); 4. fin_metodo

Tabla Nº 51. Método createMenus de la clase QAnaglyphView

createActions() **{ Método que maneja el evento de crear un menú para las diferentes opciones }**
Pre-condición: Post-condición:
1. enú = mBar->addMenu(tr(“&Archivo”)); 2. enú->addAction(newProyectAction); 3. enú->addAction(openProyectAction) 4. enú->addAction(saveProyectAction); 5. enú->addAction(saveProyectAsAction);

```
6.  enú->addAction(closeAction);
7.  enú = mBar->addMenu(tr("&Miscelaneos"));
8.  enú->addAction(parametersAction);
9.  enú->addAction(dataCollectionAction);
10. enú->addAction(doubleBufferAction);
11. Fin_metodo
```

Tabla Nº 52. Método createActions de la clase QAnaglyphView

QAWidget (parent = 0 : puntero a QWidget, f = 0 : Qt::WFlags) : QWidget (parent, f) { Constructor de la clase }
Pre-condición: Post-condición: Crea una nueva instancia del objeto
1. QAnaglyphView anaglyphView (this); 2. Q_CHECK_PTR (anaglyphView); 3. QSlider slider (Qt::Vertical,this); 4. Q_CHECK_PTR (slider); 5. Puntero a QHBoxLayout layout; 6. Q_CHECK_PTR (layout); 7. Q_CHECK_PTR(layout); 8. layout->addWidget(anaglyphView); 9. layout->addWidget(slider); 10. this->setLayout(layout); 11. connect(slider, SIGNAL(valueChanged(int)), this, SLOT(moveAnaglyph(int)));

Tabla Nº 53. Constructor de la clase QAWidget

~QAWidget () { Destructor de la clase }
Pre-condición: Post-condición: Destruye el objeto
1. Eliminar anaglyphView; 2. Eliminar slider;

Tabla Nº 54. Destructor de la clase QAWidget

init(photoNameLeft : QString, photoNameRight : QString, pos : Puntero a QPoint, zero : Puntero a QPoint, dDialog : Puntero a QDDialog, dpi : Entero, mb : Entero, pr : Entero, zr : Doble) { Método que asigna los nombres de la fotografías y los parámetros }

Pre-condición: Post-condición:
1. anaglyphView->setNameLeft(photoNameRight); 2. anaglyphView->setNameRight(photoNameLeft); 3. anaglyphView->setParameters(pr,zr,mb,dpi); 4. anaglyphView->setDDialog(dDialog); 5. anaglyphView->init(pos,zero,slider);

Tabla Nº 55. Método init de la clase QAWidget

returnPoint ( ) **{ Método que retorna el punto homologo }**
Pre-condición: Post-condición: Retorna un objeto de tipo QPoint
1. Retorna anaglyphView->returnPoint();

Tabla Nº 56. Método returnPoint de la clase QAWidget

moveAnaglyph(value : Entero) ( ) **{ Método crea el anaglifo con una nueva separación horizontal }**
Pre-condición: Post-condición: Crea el anaglifo con una nueva separación horizontal
1. anaglyphView->setZ(value/3); 2. anaglyphView->moveAnaglyph();

Tabla Nº 57. Método moveAnaglyph de la clase QAWidget

keyPressEvent(e : Puntero a QKeyEvent) **{ Método que maneja el evento "Presionar una tecla" }**
Pre-condición: Post-condición: Cierra la ventana y emite una señal formAccepted()
1. si e->key()==Qt::Key_Escape entonces 1.1. this->close(); 1.2. emit formAccepted(); 2. fin de si

Tabla Nº 58. Método keyPressEvent de la clase QAWidget

wheelEvent(e : Puntero a QWheelEvent) { Método que maneja el evento “Mover el Scroll” }
Pre-condición: Post-condición: Crea el anaglifo con una nueva separación horizontal
1. anaglyphView->setZ (anaglyphView->getZ()+e->delta()/WHEEL_DELTA); 2. anaglyphView->moveAnaglyph();

Tabla Nº 59. Método wheelEvent de la clase QAWidget

Q3CVBase(parent = 0 : Puntero a QWidget, name = 0 : Puntero a Carácter, f = 0 : Qt::WFlags) : Q3CanvasView(parent, name, f) { Constructor de la clase }
Pre-condición: Post-condición: Crea una nueva instancia del objeto
1. activeItem=NULL;

Tabla Nº 60. Constructor de la clase Q3CVBase

Q3CVBase(canvas : Puntero a Q3Canvas, parent = 0 : Puntero a QWidget, name = 0 : Puntero a Carácter, f = 0 : Qt::WFlags) : Q3CanvasView(canvas, parent, name, f) { Constructor de la clase }
Pre-condición: Post-condición: Crea una nueva instancia del objeto
1. activeItem=NULL;

Tabla Nº 61. Constructor de la clase Q3CVBase

~Q3CVBase() { Destructor de la clase }
Pre-condición: Post-condición: Destruye el objeto
1. si canvas() entonces 1.1. clear() 2. fin si

Tabla Nº 62. Destructor de la clase Q3CVBase

getActiveItem() **{ Metodo que retorna el objeto activo en el visor }**
Pre-condición: Post-condición: Retorna un objeto de tipo Q3CanvasItem
1. Retorna activeItem;

Tabla Nº 63. Método getActiveItem de la clase Q3CVBase

setActiveItem(item : Puntero a Q3CanvasItem) **{ Método que asigna un nuevo objeto activo en el visor }**
Pre-condición: Post-condición:
1. Si item != activeItem entonces 1.1. Si activeItem entonces 1.1.1. activeItem->setActive(false); 1.2. fin si 1.3. activeItem = item; 1.4. si activeItem entonces 1.4.1. activeItem->setActive(true); 1.5. fin si 1.6. canvas()->update(); 2. fin si

Tabla Nº 64. Método setActiveItem de la clase Q3CVBase

showNewItem(item : Puntero a Q3CanvasItem) **{ Método que muestra un nuevo objeto activo en el visor }**
Pre-condición: Item debe ser un puntero distinto de nulo Post-condición: Actualiza el objeto activeItem y muestra el objeto en el visor
1. setActiveItem(item); 2. item->show(); 3. canvas()->update();

Tabla Nº 65. Método showNewItem de la clase Q3CVBase

clear() { Método que elimina todos los objetos que se encuentran en el visor }
Pre-condición: Post-condición: Elimina todos los elementos que se encuentran en el visor
1. Q3CanvasItemList list = canvas()->allItems(); 2. Q3CanvasItemList::Iterator it = list.begin(); 3. Repita para hasta it != list.end() con ++it 3.1. Si it entonces 3.1.1. eliminar it; 3.1.2. fin si 4. fin repita para

Tabla Nº 66. Método clear de la clase Q3CVBase

QPhotography() { Constructor de la clase }
Pre-condición: Post-condición: Crea una nueva instancia del objeto
1. pixmap = 0;

Tabla Nº 67. Constructor de la clase QPhotography

QPhotography(datafilename : QString, format = 0 : Puntero a Carácter) { Constructor de la clase }
Pre-condición: Post-condición: Crea una nueva instancia del objeto
1. dataFileName = datafilename; 2. pixmap = new Q3CanvasPixmap(datafilename);

Tabla Nº 68. Constructor de la clase QPhotography

~QPhotography() { Destructor de la clase }
Pre-condición: Post-condición: Destruye el objeto
1. Eliminar pixmap = 0;

Tabla Nº 69. Destructor de la clase QPhotography

getDataFileName() **{ Método que retorna el nombre de la fotografía }**
Pre-condición: Post-condición: Retorna una cadena con el nombre de la fotografía
1. Retorna dataFileName;

Tabla Nº 70. Método getDataFileName de la clase QPhotography

getPixmap() **{ Método que retorna un objeto del tipo Q3CanvasPixmap }**
Pre-condición: Post-condición: Retorna un objeto del tipo Q3CanvasPixmap
1. Retorna pixmap;

Tabla Nº 71. Método getPixmap de la clase QPhotography

saveAs(datafilename : QString, format : Puntero a Carácter, quality = -1 : Entero) **{ Método que almacena la fotografía en el path descrito en "datafilename" }**
Pre-condición: Post-condición: Guarda el archivo de imagen en la ruta descrita en la cadena datafilename
1. Si datafilename == "" entonces 1.1. retornar falso; 2. fin si 3. dataFileName = datafilename; 4. Retorna pixmap->save(datafilename, format, quality);

Tabla Nº 72. Método saveAs de la clase QPhotography

loadPhoto(datafilename = -1 : QString) **{ Método que carga la fotografía }**
Pre-condición: Post-condición:

1. Si datafilename == "" entonces 1.1. retornar falso; 2. fin Si; 3. dataFileName = datafilename; 4. eliminar pixmap; 5. pixmap = new Q3CanvasPixmap(datafilename); 6. Retorna isNull();

Tabla Nº 73. Método loadPhoto de la clase QPhotography

isNull() **{ Método que verifica si la fotografía es valida }**
Pre-condición: Post-condición: Retorna un valor booleano dependiendo si la fotografía es nula o no
1. Retorna pixmap->isNull();

Tabla Nº 74. Método isNull de la clase QPhotography

width() **{ Método que retorna un entero que representa el ancho de la fotografía }**
Pre-condición: Post-condición: Retorna el ancho de la fotografía
1. Retorna pixmap->width();

Tabla Nº 75. Método width de la clase QPhotography

height() **{ Método que retorna un entero que representa el largo de la fotografía }**
Pre-condición: Post-condición: Retorna el largo de la fotografía
1. Retorna pixmap->height();

Tabla Nº 76. Método height de la clase QPhotography

QPoint3D(_x: doble, _y : doble, _z : doble) : QPointF(_x, _y) **{ Constructor de la clase }**

Pre-condición: Post-condición: Crea una nueva instancia del objeto
1. Z = _z;

Tabla Nº 77. Constructor de la clase QPoint3D

QPoint3D() **{ Constructor de la clase }**
Pre-condición: Post-condición: Crea una nueva instancia del objeto
1. QPoint3D(0,0,0);

Tabla Nº 78. Constructor de la clase QPoint3D

operator=(p3d : constante QPoint3D) **{ Sobrecarga del operador = }**
Pre-condición: Post-condición:
1. setX(p3d.x()); 2. setY(p3d.y()); 3. Z = p3d.Z; 4. Retorna *(new QPoint3D(x(),y(),Z));

Tabla Nº 79. Sobrecarga del operador = de la clase QPoint3D

~QPoint3D() **{ Destructor de la clase }**
Pre-condición: Post-condición: Crea una nueva instancia del objeto

Tabla Nº 80. Destructor de la clase QPoint3D