universidad del país vasco — euskal herriko unibertsitatea

Facultad de Informática — **Informatika Fakultatea**

TITULACIÓN:

Ingeniería Informática

Herramienta de Autor para la Creación de 3D Trackable Model (3DTM)

Alumno: Rubén de Celis Hernández

Director: Alejandro García-Alonso Montoya

Proyecto Fin de Carrera, 1 de febrero de 2014

Agradecimientos

En primer lugar me gustaría expresar mi agradecimiento a todas aquellas personas que me han apoyado para poder completar este trabajo:

A mi director de proyecto Alex García-Alonso por todo el esfuerzo y el tiempo dedicado, siempre orientándome durante el desarrollo del proyecto. Es un placer haber podido desarrollar mi proyecto bajo su diligente dirección.

Agradecimientos a Jairo Roberto Sánchez, mi tutor, quien me ha guiado y ayudado durante todo el proyecto, resolviendo mis dudas y ofreciéndome siempre las mejores explicaciones.

Mención también para David Oyarzun Laura, director del área de Animación 3D y Entornos Virtuales Interactivos, por darme la oportunidad de realizar mi PFC en Vicomtech.

No quisiera olvidarme de mis compañeros, y en especial de Nagore, Víctor y Javier, que además de ser buenos profesionales, también son estupendas personas.

Mención aparte para mi familia quien me ha apoyado siempre y que seguramente que sin su apoyo incondicional no habría llegado hasta donde estoy. Gracias por no dejar que nunca me conforme y haberme animado siempre a seguir.

RESUMEN

El presente proyecto describe una herramienta de autor que facilita el *tracking* sin marcadores de objetos 3D con textura. La herramienta está orientada a la detección de posición mediante un sistema mono-cámara para entornos en los que se tienen modelos creados, como suele ser usual en los proyectos de arquitectura y de ingeniería.

La finalidad de la herramienta es la generación de un 3DTM que contiene los puntos de interés del objeto, sus descriptores y sus correspondientes puntos 3D en el modelo, obtenidos a partir de una imagen y un modelo del objeto. De esta forma el 3DTM generado, puede ser usado para el *tracking* de objetos 3D con textura en aplicaciones de Realidad Aumentada.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 INTRODUCCIÓN

Vicomtech-IK4 es un centro de investigación aplicada especializado en las tecnologías de Computer Graphics, Visual Computing y Multimedia. Desde hace unos años Vicomtech añadió entre sus líneas de investigación la Realidad Aumentada (RA).

Vicomtech desde entonces ha desarrollado software que ha dado lugar a una librería propia que utiliza intensivamente en sus investigaciones y desarrollos. La librería permite *tracking* eficiente de marcadores planos con textura de la propia escena, en equipos informáticos y dispositivos móviles.

Como evolución natural la librería se adaptó para funcionar con una arquitectura de cliente servidor en la red. Esta característica posibilitó poder llevar los cálculos más costosos del proceso de *tracking* a servidores con la potencia suficiente, para disminuir los tiempos de procesado que se producían en los dispositivos móviles. De esta forma la librería empezó a hacer uso de la técnica de computación en la nube.

Por otro lado Vicomtech también desarrolla herramientas de autor que facilitan el uso de la tecnología RA a usuarios finales. Las herramientas principalmente se basan en la Web3D y la tecnología móvil. Podemos encontrar un ejemplo en la publicación de Barbadillo [BS13] sobre una herramienta de autor para RA basada en Web3D.

El presente proyecto surge de una necesidad por parte de Vicomtech. Se necesitaba una herramienta que facilitase el *tracking* de objetos 3D con textura. La herramienta se añadirá a la suite de software de RA de Vicomtech.

El prototipo desarrollado en este proyecto se presenta como una herramienta de autor para la creación de lo que se ha acuñado como *3D Trackable Model*, abreviado como 3DTM. La herramienta genera el 3DTM con un mínimo esfuerzo para el usuario. El 3DTM se podrá usar en aplicaciones de RA para el *tracking* de objetos 3D con textura.

La documentación recogida en la presente memoria, describe las técnicas utilizadas durante el desarrollo de la herramienta y no las particularidades de la implementación. La descripción técnica de la implementación se genera con Doxygen desde el código fuente.

En el siguiente subapartado se recogen algunas precisiones terminológicas. Los siguientes subapartados presentan los objetivos del proyecto, la descripción de tareas y el diagrama de Gantt.

El resto de la memoria está constituida por un capítulo dedicado a antecedentes, otro que describe las tareas realizadas y la bibliografía.

Se incluyen los siguientes anexos:

- Especificación del fichero 3DTM
- Sistema de carpetas y ficheros de la Herramienta RProject
- Planificación y Modelo de Desarrollo
- Manual de Usuario de la Herramienta RProject
- Doxygen: Propuesta de estilo para Vicomtech

1.1 Terminología

Los siguientes términos se usarán en este documento siguiendo las descripciones que se realizan a continuación:

Objeto: Una entidad real del mundo físico. Puede ser una pieza fabricada, una estatua, un edificio, etc…

Modelo: La representación geométrica de un objeto mediante polígonos (vértices -geometría- y polígonos -topología-). Sería ideal que los polígonos constituyan un poliedro, pero no se considera necesario en este caso.

Imagen: *Fotografía* de un objeto o *modelo proyectado* (renderizado de un modelo). Cuál de los dos casos se considera en una frase dada, normalmente quedará determinado por el contexto. En ocasiones para evitar imprecisiones se usará en vez de imagen el término ***fotografía*** o ***modelo proyectado***.

Parámetros (de una imagen): Definen las matrices de rotación y traslación que determinan la posición relativa de la cámara respecto al objeto en el momento de obtener la fotografía. También definen esas matrices en el caso de posicionar una cámara virtual respecto a un modelo. En el caso de una fotografía se necesita haber definido previamente un sistema de referencia del mundo. En el caso de una imagen proyectada se usará el sistema de referencia del modelo (en el que se expresan las coordenadas de los vértices del modelo).

Renderizar: Término usado en para referirse al proceso de generar una imagen o vídeo mediante el cálculo de iluminación partiendo de un modelo en 3D.

Envolvente: Polígono de color homogéneo que delimita la silueta de un poliedro una vez fijada su proyección.

Máscara: Es una entidad que controla el comportamiento de una colección de píxeles. En términos generales, este control se centra en la relativa transparencia u opacidad de los píxeles.

Keypoint: Término que se utiliza en Realidad Aumentada para referirse a los puntos de interés obtenidos tras el uso de algún algoritmo como FAST [RD06]. Los *keypoints* son puntos que por su gradiente de color, son fácilmente diferenciables de los de su entorno.

Unprojection: Procedimiento que permite convertir coordenadas de pantalla en puntos 3D del espacio. Para ello se busca la intersección de rayo proyectado desde las coordenadas de pantalla y un objeto en el espacio 3D.

YAML: Es un formato de serialización de datos legible por humanos inspirado en lenguajes como XML.

Matching: Término usado para referirse al proceso de emparejamiento entre elementos de dos grupos. En Realidad Aumentada el proceso se concreta en el emparejamiento de *keypoints* de dos imágenes atendiendo a sus descriptores.

Distancia: El parámetro de distancia define la similitud existente entre los descriptores de un par de *keypoints* en un emparejamiento. Cuanto mayor es la similitud, menor es la distancia.

1.2 Objetivos

El objetivo de este proyecto es la creación de una herramienta de Realidad Aumentada (RA) sin marcadores artificiales, que permita la creación de *3D Trackable Models* (3DTM).

La herramienta será un prototipo funcional. Estará enfocada como una herramienta de autor para que cualquier usuario, sin conocimientos en RA, pueda utilizarla.

Los hitos contemplados para implementar esta herramienta de autor son:

- Creación de un sistema de emparejamiento manual entre el modelo de un objeto y una imagen del mismo.
- Delimitación del objeto en la imagen.
- Aislamiento del objeto de su entorno en la imagen.
- Extracción de puntos de interés que permitan detectar el objeto.
- *Unprojection* de los puntos de interés del objeto en 2D para conseguir puntos en el modelo en 3D.
- Creación del 3DTM.
- Evaluación del 3DTM realizando *matchings* contra un conjunto nuevo de imágenes del objeto.

1.3 Descripción de Tareas

1 Emparejamiento manual de puntos imagen con vértices del modelo

Sea un objeto, se dispone de su modelo y de una fotografía del mismo. En una ventana se mostrará esa imagen, en otra ventana se proyecta el modelo pudiendo el usuario cambiar el punto de vista. El usuario creará una colección mínima de cuatro pares de puntos. Para ello, se seleccionará un vértice del modelo y su correspondiente punto en la imagen. El punto del modelo se puede seleccionar usando cualquier proyección y será aproximado al vértice más cercano. Cada punto del modelo puede seleccionarse en una proyección distinta del modelo.

2 Cálculo de los parámetros de una imagen y proyección de vértices del modelo en esa imagen

Dada la colección de pares de puntos obtenidos en la tarea previa de emparejamiento de puntos, se resuelve el problema PNP (Perspective N Point) para conseguir los valores de rotación y traslación del sistema de referencia de la cámara respecto al objeto (usando el sistema de referencia del mundo). Estos valores definen la posición relativa de la cámara respecto al objeto. Utilizando los parámetros calculados (rotación y traslación) se proyectan los vértices del modelo en la imagen. De esta forma los vértices proyectados del modelo quedan superpuestos con sus correspondientes vértices del objeto en la imagen.

3 Cálculo de la envolvente del modelo proyectado

Dados el modelo, la imagen y los puntos proyectados obtenidos del paso anterior, se dibuja sobre la imagen el poliedro que define la silueta del modelo proyectado. Esto permite la posterior delimitación en la imagen del objeto respecto a su entorno.

4 Generación de máscara de delimitación del modelo proyectado

Dada la imagen en la que se ha dibujado la envolvente, se genera una *máscara* que permite filtrar sólo aquella sección de la imagen original que corresponde al objeto. En este paso se consigue aislar el objeto del resto de la imagen mediante la aplicación de filtros de color. La región o regiones de la imagen no correspondientes al objeto y que quedan fuera de la máscara, son rellenadas de un color homogéneo.

5 Obtención de *Keypoints* y Descriptores del objeto

Dada la imagen con el objeto aislado de su entorno, se extraen sus puntos de interés (*keypoints*) usando el algoritmo FAST y sus correspondientes descriptores usando FREAK. La utilización de FAST y FREAK para la detección de *keypoints* y extracción de los descriptores de los mismos, es un ejemplo ilustrativo para el funcionamiento de la herramienta, pudiendo ser utilizados otros algoritmos.

6 *Unprojection*

Dados los *keypoints* obtenidos de la imagen 2D, se buscan los puntos 3D correspondientes en la superficie del modelo. En este caso los puntos en el modelo no son vértices necesariamente, sino punto situados en el interior o frontera del alguno de sus polígonos.

7 Guardado del Modelo de *Tracking*

Se genera el *3D trackable model* y se guarda en un fichero YAML.

Se denomina *3D trackable model* al conjunto de características que describen la apariencia y correspondencia de un objeto con su modelo 3D.

Las características principales que describen un *3D trackable model*:

- *Keypoints*: Puntos de interés 2D del objeto extraídos de una imagen.
- *Descriptors*: Descriptores que contienen información identificativa de la región dónde se encuentra cada *Keypoint*. Permite identificar el punto en una nueva imagen.
- *3D Points*: Coordenadas 3D de los *keypoints*. Corresponden a la superficie del modelo.

8 Detección de *Keypoints* del Objeto en una nueva imagen

Dados el 3D trackable model y una nueva imagen en la que aparece el objeto, se buscan los *keypoints* en la nueva imagen y sus correspondientes descriptores. Mediante un algoritmo de *matching* de fuerza bruta, se realizan los emparejamientos entre los *keypoints* del objeto en la nueva imagen y los conocidos del objeto. Este *matching* se realiza atendiendo a la similitud de los descriptores. Después de ejecutar el algoritmo, se filtran aquellos pares de *keypoints* que presentan una mejor precisión en el emparejamiento atendiendo al valor de *distancia*. Del mismo modo se filtran los puntos 3D del modelo y los descriptores correspondientes a los *keypoints*.

De nuevo se trata de un ejemplo ilustrativo para el funcionamiento de la herramienta, pudiendo ser utilizado otro algoritmo de *matching*.

9 Cálculo iterativo de los parámetros de una imagen y proyección del modelo en esa imagen

Dados los mejores *keypoints*, junto con los puntos 3D del modelo y descriptores correspondientes a los *keypoints*. Se resuelve el problema PNP (Perspective N Point) para conseguir los parámetros de la imagen. La resolución esta vez del problema PNP se realiza mediante un proceso iterativo en busca de la mejor solución usando los *keypoints* y no los vértices como en la *Tarea 2*.

Utilizando los parámetros de la imagen (rotación y traslación) se proyectan los vértices del modelo en la imagen. De esta forma los vértices proyectados del modelo quedan superpuestos con sus correspondientes vértices del objeto en la imagen.

1.4 Diagrama de Gantt

En la Figura 1 y la Figura 2 se muestra un resumen de la planificación del proyecto que abarca desde el 16 de septiembre de 2013 hasta el 31 de enero del 2014. Las jornadas son de 8 horas.

En el diagrama de Gantt (véase Figura 1), se muestra la duración de las tareas y sus fechas de inicio y de finalización.

En el gráfico de Gantt (véase Figura 2), se pueden apreciar más claramente las dependencias entre las distintas tareas.

	Nombre	Duración	Inicio	Fin	Predecesoras
0	⊟__Planificación RProject	85.63d	16/09/2013	31/01/2014	
1	⊞Formación	7d	16/09/2013	24/09/2013	
4	⊟Tareas	58.38d	24/09/2013	18/12/2013	1
5	⊞Tarea 1	9d	24/09/2013	07/10/2013	
11	⊞Tarea 2	12d	07/10/2013	23/10/2013	5
17	⊞Tarea 3	3.88d	23/10/2013	30/10/2013	11
23	⊞Tarea 4	2.63d	30/10/2013	04/11/2013	17
29	⊞Tarea 5	6.25d	04/11/2013	13/11/2013	23
35	⊞Tarea 6	9.5d	13/11/2013	26/11/2013	29
41	⊞Tarea 7	6d	26/11/2013	04/12/2013	35
47	⊞Tarea 8	2.25d	04/12/2013	09/12/2013	41
53	⊞Tarea 9	6.88d	09/12/2013	18/12/2013	47
59	⊞Interfaz	2d	18/12/2013	20/12/2013	4
65	Documentación	17d	07/01/2014	31/01/2014	

Figura 1: Diagrama Gantt del Proyecto

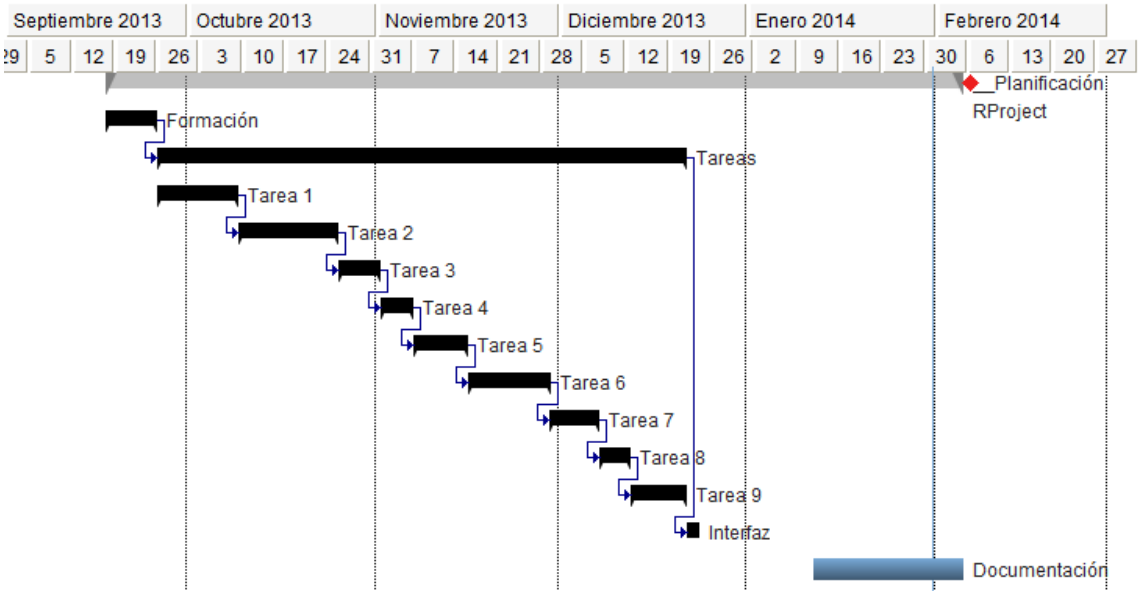


Figura 2: Gráfico Gantt del Proyecto

2. ANTECEDENTES

En este apartado se presenta un resumen del estado del arte en la Realidad Aumentada para situar al lector y contextualizar los contenidos que se desarrollen más adelante. En segundo lugar se presenta también brevemente los elementos de la arquitectura de la herramienta.

2.1 ¿Qué es la Realidad Aumentada?

La Realidad Aumentada (RA) se define como la tecnología capaz de enriquecer la realidad percibida con información generada por ordenador, de tal forma que no se distinga la información percibida del mundo real de la generada por un sistema de RA. Esta característica permite que se puedan mezclar objetos virtuales con la realidad percibida, permitiendo crear la ilusión de que los objetos virtuales coexisten en el mismo mundo con los reales (véase Figura 3).

Figura 3: Tetera virtual en un entorno real

Más formalmente Azuma [A97] definió en 1997 la RA de la siguiente forma:

- Combina elementos e información virtuales y reales
- Es interactiva en tiempo real
- Está registrada en 3D

El trabajo de Milgram y Kishino [MK94] define RA como un continuo que abarca desde el entorno real a un entorno virtual puro. Entre medio hay Realidad Aumentada más cerca del entorno real y Virtualidad Aumentada que está más cerca del entorno virtual (véase Figura 4).

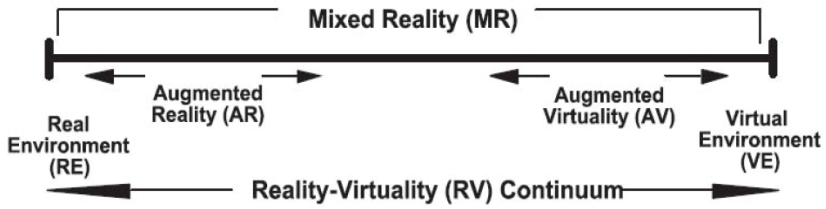


Figura 4: Reality-Virtuality Continuum

El usuario final de un sistema de RA, contempla una imagen en donde estará proyectado el mundo (que ha sido capturado a través de una cámara) y la proyección de uno o varios elemento 3D, en dicha imagen. El objetivo, es colocar elementos virtuales en posiciones concretas del mundo. Para ello, se debe identificar la localización de un objeto/lugar continuamente, incluso cuando el objeto o la propia cámara se están moviendo. A este proceso se le denomina *tracking*.

2.2 Técnicas de Localización

El objetivo primero de un sistema de RA, es la localización en tiempo real del usuario en el entorno, o lo que es lo mismo en este caso, de la cámara. Existen en el estado del arte, diferentes métodos para alcanzar dicho objetivo (véase Figura 5).

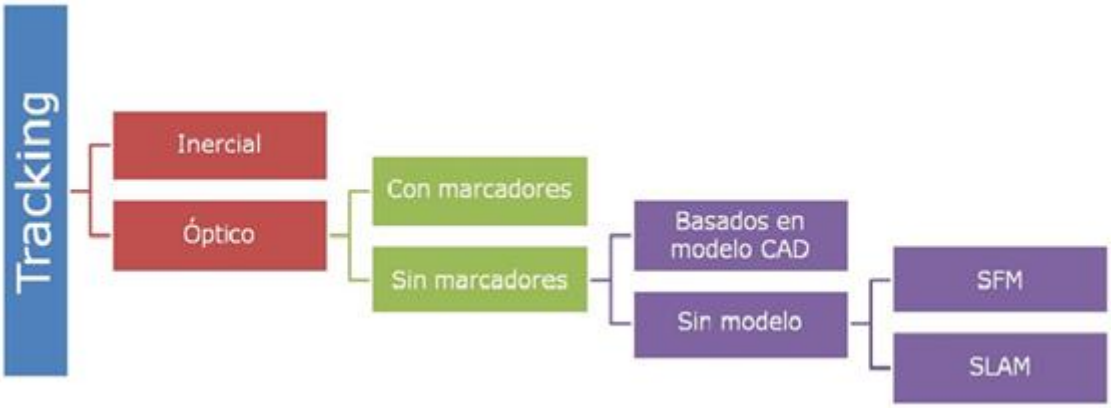


Figura 5: Clasificación de métodos de tracking en RA

Los dos tipos de *tracking* principales son el inercial y el óptico, siendo el segundo el que más ha avanzado y más investigación ha tenido en los últimos años. Esto es debido a que resulta más preciso que el *tracking* inercial.

El *tracking* óptico se basa en la estimación exacta de la posición y orientación de la cámara en tiempo real partiendo de las imágenes capturadas por la propia cámara. Cuando al entorno debe añadirse algún tipo de infraestructura para poder hacer dicha estimación, entonces hablamos de *tracking* con marcadores. Pero, por lo contrario, si ningún tipo de infraestructura debe ser añadida en el entorno, hablamos de *tracking* sin marcadores.

2.2.1 *Tracking* Inercial

En este método se aprovecha la información que puedan aportar los sensores inerciales del dispositivo con el que se vaya a realizar una aplicación de RA, para obtener la localización del objeto en movimiento en el mundo real. Los sistemas inerciales son totalmente independientes de métodos externos y calculan la localización a través de la aceleración lineal y el movimiento angular.

Un sensor inercial está compuesto por acelerómetros, giróscopos y magnetómetros. Los acelerómetros medirán la aceleración lineal con la que se mueve el sensor, los giróscopos la velocidad angular y los magnetómetros dan información sobre el norte magnético. Este método calcula la posición de un objeto en movimiento, la cámara en nuestro caso, relativa a una posición inicial. Es decir, el usuario debe inicializar el sistema con la posición inicial de la cámara, ya que el sistema es incapaz de inicializarla.

Es inherente a este método que el error se vaya incrementado a lo largo del tiempo. A esta característica se le denomina *drift*. La única alternativa para solucionar este problema, es volver a inicializar el sistema con la nueva posición de la cámara cada cierto tiempo. Un sistema externo, que calcule la posición de la cámara cada cierto tiempo, puede paliar de alguna manera dicho problema. Es importante tener presente el compromiso entre la tolerancia al error acumulado y las veces que se reinicializa el sistema.

Existen en el mercado sistemas de RA para dispositivos móviles basados en *tracking* inercial. Layar, por ejemplo, es una aplicación de RA para teléfonos móviles con sistema operativo Android (véase Figura 6). En este caso, además de los acelerómetros y giróscopos se hace uso del GPS y la brújula para hacer el *tracking*.

Figura 6: Aplicación Layar

Layar recupera los datos basados en coordenadas geográficas y los superpone sobre la vista de la cámara. Cuando se utilizan sensores inerciales (acelerómetros y brújulas) y GPS para rastrear el dispositivo, la información mostrada normalmente se limita a etiquetas 2D. La exactitud de las lecturas de los sensores no es suficiente para insertar contenido 3D correctamente alineado con imágenes de la cámara.

Otra aplicación similar, es la de Wikitude (véase Figura 7). Una aplicación de RA para móviles Android, que añade información adicional a elementos reales, tales como edificios, monumentos, etc. Es una aplicación usada sobre todo en el ámbito del turismo. Esta aplicación al igual que Layar, hace uso del *tracking* inercial, complementándolo con la información del GPS y la brújula.

Figura 7: Aplicación Wikitude

El método de *tracking* inercial sigue siendo uno de los más utilizados por los desarrolladores de aplicaciones de RA para dispositivos móviles. Han sido numerosas las aplicaciones que hacen uso de los sensores inerciales para localizar el dispositivo.

Entre esas aplicaciones está Dish Pointer (véase Figura 8), una aplicación diseñada para ejecutarse en tablets y smartphones de Apple, que mediante un simple modelo representa la localización de los satélites más cercanos a la ubicación del usuario. Esta aplicación hace uso de la información que aporta el GPS del dispositivo para localizarse, y al dirigir este último hacia el cielo, se sobreponen imágenes 2D que representan la ubicación de los satélites más cercanos.

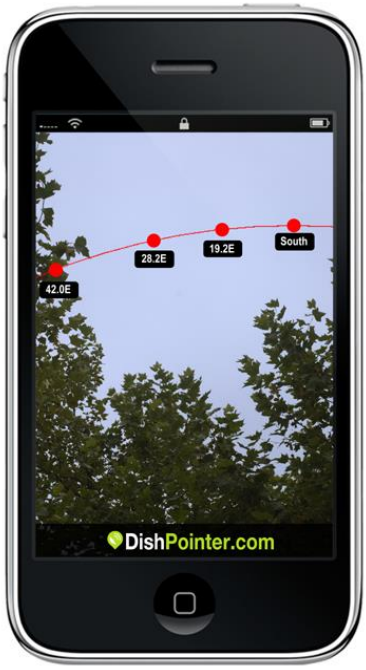


Figura 8: Aplicación Dish Pointer

Otra aplicación relevante es Car Finder AR. Es una aplicación que con la ayuda de mapas y RA, guía al usuario hacia el lugar en donde estacionó su vehículo. En la Figura 9 se puede observar la aplicación, tanto cuando usa los mapas 3D de Google, como haciendo uso de la RA.

Figura 9: Aplicación Car Finder AR

2.2.2 *Tracking* con Marcadores

Los sistemas de *tracking* basados en marcadores, usan marcas que pueden ser reconocidas, normalmente marcas impresas, que serán colocadas en el entorno. Estas marcas serán capturadas por la cámara y reconocidas, gracias a un entrenamiento previo. Las marcas facilitan que la estimación de la posición de la cámara sea una tarea más sencilla. En la Figura 10, se puede observar un ejemplo de marcador.

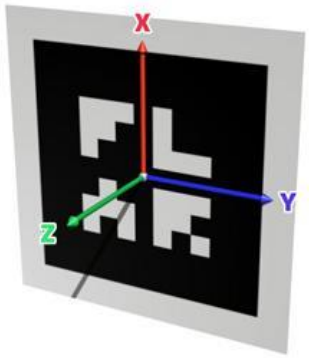


Figura 10: Marcador

Como el marcador es un elemento conocido, que ha sido entrenado previamente, es muy sencillo conseguir su posición con gran precisión. De tal manera, que los marcadores se añadirán en aquellos lugares del mundo real en el que se desee añadir los elementos virtuales. Es decir, el modelo 3D siempre se tendrá que posicionar en la cercanía de los marcadores (véase Figura 11).

Figura 11: Ejemplo de tracking con marcador

Este sistema de *tracking* es sencillo de implementar y muy preciso, pero se debe intervenir en la escena para que una aplicación de RA pueda funcionar correctamente, lo que hace que sea un sistema poco práctico para algunas aplicaciones.

Existen en el mercado infinidad de aplicaciones de RA que realizan *tracking* basado en marcadores. Una de las aplicaciones más conocidas es el juego Invizimals de PSP (véase Figura 12). El lugar de juego es el entorno que el usuario elija, el propio entorno que lo rodea. El resto de elementos del juego, personajes, objetos, etc. son elementos virtuales que se añadirán al entorno real escogido. Para ello, hace uso de la propia cámara de la PSP y de marcadores colocados en lugares del entorno, donde serán superpuestos los elementos virtuales del juego.

Figura 12: Imagen del juego Invizimals

Un ejemplo de uso muy extendido son los llamados marcadores ARToolkit. Éste es un conjunto de librerías para el desarrollo de aplicaciones de RA. Usa marcadores para localizar el dispositivo en todo momento. Estos marcadores son bastante característicos y en los ejemplos hasta ahora mencionados hemos podido ver distintas aplicaciones que usan esta librería y sus marcadores.

La manera de no intervenir en la escena pero usar marcadores para la localización, se consigue mediante el uso de los denominados marcadores naturales. El método se basa en usar marcas conocidas y previamente entrenadas, para estimar la posición de la cámara con precisión y de manera relativamente sencilla. En este caso, en vez de ser marcas como las mencionadas hasta ahora, serán elementos ya existentes en el entorno. De esta manera, no se interviene en la escena ni se adultera. Los elementos previamente entrenados y usados para hacer el *tracking* son elementos que ya existen en la escena y en cambio el *tracking* sigue siendo igualmente preciso y robusto que con los marcadores de ARToolkit.

2.3.3 *Tracking* sin Marcadores

En el *tracking* sin marcadores cualquier parte del entorno real puede ser utilizado como un marcador, ya que el sistema explota las características naturales presentes en la escena real para realizar el seguimiento [BPG10].

Un ejemplo de éxito es Junaio (véase Figura 13), un navegador de RA para dispositivos móviles (sistema operativo IOS o Android). Con Junaio, el usuario puede escoger la imagen que desee y subirla a la aplicación. Esta imagen servirá de marcador natural para la aplicación de RA que el propio usuario quiera diseñar sin tener la necesidad de generar ni una sola línea de código.

Es bastante conocida la aplicación de marketing que ofrece Junaio sobre los juguetes Lego que cobran vida gracias a la RA. En esta aplicación, la propia imagen de la caja del juguete es usada como marcador, sin haber tenido que añadir ningún elemento al entorno. Lo mismo ocurre con la famosa aplicación que facilita este navegador, sobre algunas revistas con RA. Las propias páginas de la revista son el marcador.

Figura 13: Aplicación Junaio

Existen diferentes técnicas para desarrollar el *tracking* sin marcadores. Dichas técnicas se pueden clasificar en los siguientes grupos: los basados en modelos (Model-based), los basados en estructuras de movimiento, en adelante SfM (Structure for Motion) y la técnica de SLAM (Simultaneous localization and mapping).

Basados en modelos

En la técnica basada en modelos, la información del entorno real es obtenida antes del proceso de *tracking* y es guardada en un modelo 3D que será usado para estimar la posición de la cámara [VLF04]. En esta técnica la posición de la cámara es estimada al emparejar el modelo 3D de un objeto con el objeto del mundo real, basándose en la información de las aristas o texturas de éste último.

Sin modelos

En las técnicas no basadas en modelos se encuentran SfM y SLAM. La diferencia entre ambas técnicas, SfM y SLAM, radica en que en la primera es necesaria una secuencia de vídeo completa para hacer la reconstrucción del entorno, mientras que en la segunda el mapa reconstruido se va ampliando en cada fotograma capturado.

SfM

En la técnica de SfM [D00], la estimación de la posición de la cámara en cada instante, se calcula sin ningún tipo de conocimiento previo sobre el entorno, dado que éste es adquirido durante el mismo proceso de *tracking*. Es decir, se generará una estructura de imágenes 2D que representa un entorno 3D, a partir de un número de imágenes capturadas por la cámara. Para que esta técnica se pueda desarrollar las correspondencias entre la imagen y la reconstrucción del objeto 3D deben hallarse. Con el fin de hallar dichas correspondencias entre imágenes, características propias de la imagen (puntos y/o bordes), deben de ser rastreadas de una imagen a la siguiente. La trayectoria de las características a lo largo del tiempo se usa para reconstruir las posiciones en 3D y el movimiento de la cámara.

SLAM

La localización simultánea y mapeo, SLAM, es una técnica utilizada para construir un mapa de un entorno desconocido (sin un conocimiento previo), o para actualizar un mapa de un entorno conocido (con un conocimiento previo del entorno, con un mapa dado), mientras que al mismo tiempo, se calcula la posición del dispositivo utilizado para aplicar dicha técnica en el mapa que se está construyendo.

En SLAM para obtener la información del entorno se usan diversas fuentes, siendo los más comunes los sensores como: láseres, sonares, etc. Sin embargo, un método que está teniendo auge en los últimos años es la utilización de cámaras para la captura de este entorno. Esta área de investigación es conocida como Visual SLAM o VSLAM [K05] [BSG13].

2.3 Clasificación de Lepetit y Fua

La clasificación que ofrecen Lepetit y Fua [LF05] realiza una catalogación de los métodos de *tracking* 3D atendiendo a las siguientes características:

- La situación en la que se adecua el método.
- Si se requiere inicialización.
- Si es preciso.
- Los posibles casos en los que hay fallo.
- Si se requiere modelo.

La Tabla 1 presenta un resumen de su propuesta.

Basados en	Adecuado cuando	Inicialización manual	Precisión	Modos de fallo	Requiere modelo 3D
Marcadores retro-reflectantes, cámaras infrarrojas	Uso de marcadores es aceptable; No importa el gasto	No	Alta precisión	Muy raro que falle	Puede ser reconstruido por el sistema
Marcadores planos	Uso de marcadores es aceptable	No	Preciso	Ocultación del marcador	No
Aristas	Aristas fuertemente definidas, fondo simple	Sí	Temblores	Movimiento muy rápido; Fondo desordenado	Sí
Template matching	Objetos pequeños y planos	Sí	Alta precisión	Oclusiones; Movimiento muy rápido	No
Puntos de interés	Objetos planos con textura	Sí	Puede haber desviaciones	Movimiento muy rápido	No
Puntos de interés	Objetos 3D con textura	Sí	Usando keyframes se reduce el desvío y los temblores	Movimiento muy rápido	Sí
Puntos de interés	Escenas 3D con textura	No	Desvío limitado	Movimiento muy rápido	No

Tabla 1: Clasificación de métodos de tracking 3D

La Tabla 1 expone métodos de *tracking* 3D que actualmente se pueden ejecutar en tiempo real en dispositivos móviles, tales como tablets y smartphones.

Los métodos que no requieren inicialización coinciden con aquellos que necesitan marcadores artificiales que hay que introducir en la escena. Tienen como ventaja que no necesitan un modelo 3D y que incluso es posible reconstruirlo. Además no requieren de una inicialización previa para su funcionamiento.

Esta clasificación está desarrollada con mayor detalle en el trabajo "*Monocular model-based 3D tracking of rigid objects*" [LF05].

Actualmente están experimentando un auge considerable aquellos métodos que explotan las características de los elementos presentes en la escena: Esto es debido a que no necesita de elementos artificiales que hay que introducir en la escena. Los productos comerciales que se pueden encontrar actualmente, se basan en objetos planos con textura de la propia escena para realizar el *tracking* [JD02]. El siguiente paso es el uso de objetos 3D con textura para realizar el *tracking* [VLF04].

Por tanto, el enfoque más genérico y deseable es desarrollar métodos puramente basados en la imagen que puedan detectar los objetos y calcular su posición 3D a partir de una sola imagen. Además es deseable que se pueda hacer utilizando marcadores naturales de la propia escena, en vez de marcadores artificiales introducidos en la escena.

2.2 Elementos de la Arquitectura

El mapa conceptual de esta herramienta de autor se presenta en la Figura 14, es una visión general y simplificada de la arquitectura de la herramienta. Se puede acceder a la documentación incluida en el código por medio de Doxygen.

RMain es la clase que hace de controladora de todo el proceso de ejecución de la herramienta. Realiza las llamadas a las funciones del resto de las clases y marca la continuación de la ejecución tras la respuesta de cada una de las funciones de las clases. Se encarga de servir los datos de entrada para las distintas llamadas a las funciones y recoger el output que devuelven. También es la encargada de generar el fichero 3DTM en formato YAML y de empaquetar los resultados finales en un único fichero de salida en formato ZIP.

La clase RMain contiene el main para la ejecución de la herramienta. La clase RMain no dispone de interfaz integrada. Se ha desarrollado un Frontend a modo de Interfaz que facilita la entrada de datos y la interacción con la herramienta.

La herramienta tiene una diseño modular, en el que cada una de las clases principales (véase Figura 14), ofrece una serie de funciones de acuerdo a la librería que está usando.

La clase PLYLoader usa la librería PCL en su versión 1.6 para la carga de modelos 3D y manipulación de los mismos. Se utilizada principalmente al inicio al cargar un modelo y al gestionar el renderizado del modelo 3D durante la *Tarea 1*.

La clase GLUTViewer usa la librería GLUT en su versión 3.7.6 para el renderizado de modelos 3D. La clase renderiza un modelo aplicándole una vista para orientarlo y posicionarlo de una forma determinada. Además es clave para el proceso de *unprojection* de puntos 2D.

La clase IMGLoader usa la librería OpenCV en su versión 2.4.6 para la carga de imágenes, el procesamiento de las mismas y el dibujado 2D. Esta clase también puede renderizar el modelo 3D. Se usa para renderizados superpuestos a la imagen.

Otras funcionalidades de estas clases irán apareciendo en los apartados que describen la herramienta.

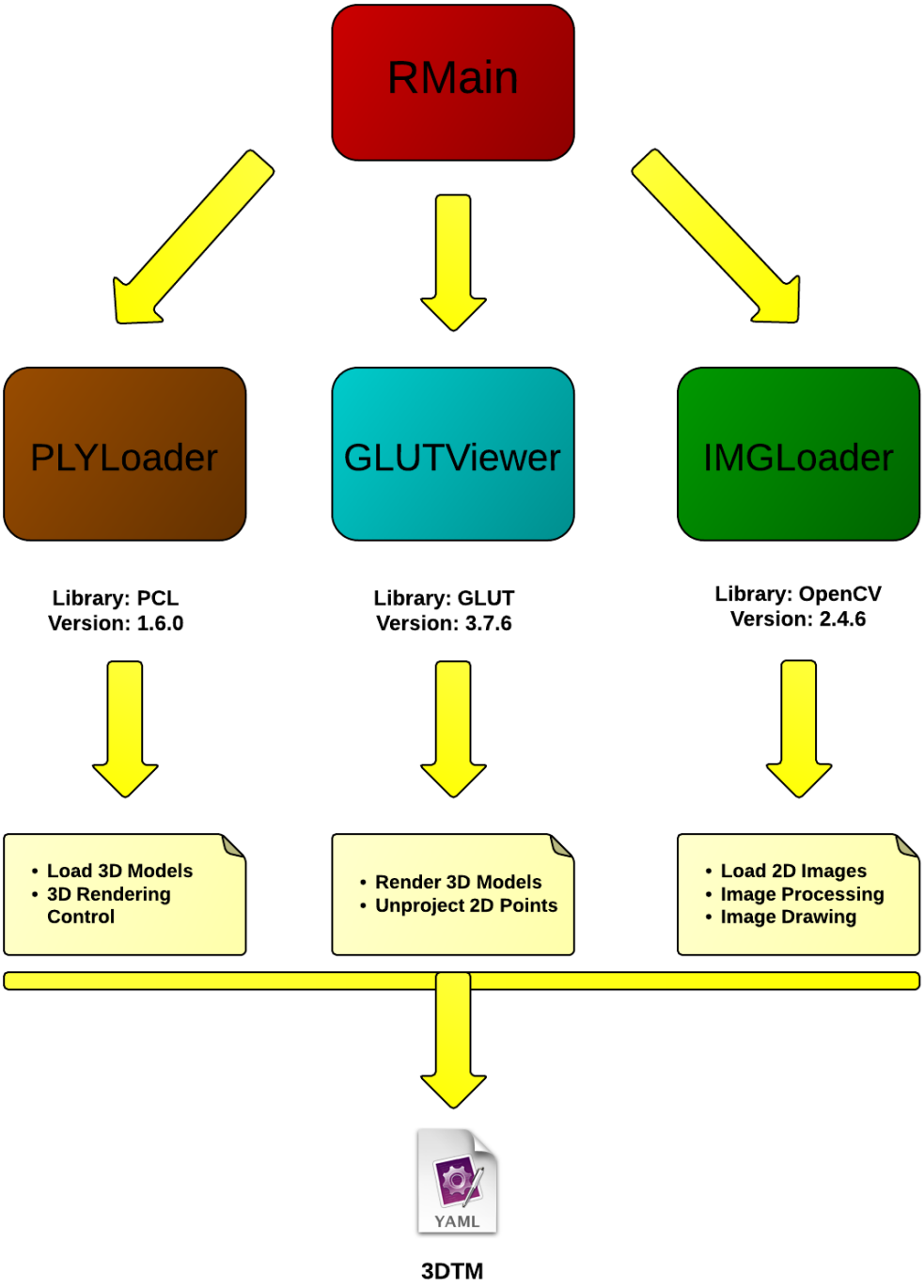


Figura 14: Arquitectura de RProject

3 TAREAS

En este apartado se describen con más detalle cada una de las tareas presentadas en la introducción. Algunas tareas añaden terminología específica que se presenta al inicio del desarrollo de la tarea.

3.1 Emparejamiento manual de puntos imagen con vértices del modelo

Terminología

- **Solid rendering**: Término que se utiliza en Gráficos por Computador para referirse a la generación de una imagen construida con superficies. En nuestro caso sería fruto de renderizar los polígonos.
- **Wired rendering**: Término que se utiliza en Gráficos por Computador para referirse a la generación de una imagen construida con aristas. En nuestro caso sería fruto de renderizar solamente la frontera de los polígonos.

Para el emparejamiento manual de puntos de correspondencia entre una imagen y un modelo, la ejecución principal lanza dos ventanas independientes que son ejecutadas concurrentemente en hilos de ejecución distintos. Esto permite la interacción concurrente con las dos nuevas ventanas. En una ventana aparece la imagen, en la otra se renderiza el modelo (véase Figura 17).

La ejecución principal corresponde a RMain, mientras que funciones de apertura de las ventanas corresponden a PLYLoader e IMGLoader (ver Sección 2.2).

La selección de la imagen y el modelo puede realizarse de dos formas:

- Al ejecutar el programa, mediante paso de parámetros con los nombres de los ficheros de imagen y de modelo a cargar.
- Al arrancar el programa sin especificar parámetros y antes de crear las ventanas, se mostrarán dos listas de las posibles imágenes y modelos que pueden ser cargados. La selección del fichero se pedirá por parte del programa y el usuario deberá escribir el nombre del fichero (incluida su extensión) que quiera cargar. Si el nombre del fichero es incorrecto se volverá a pedir de nuevo.

Los archivos son cargados desde la carpeta “DATA” (véase Anexo II). Por tanto, es necesario introducir en esa carpeta las imágenes y modelos con los que se van a trabajar.

La ventana que presenta la imagen se dibuja por pantalla usando la librería de Visión por Computador OpenCV (véase Figura 15).

Se aconseja cargar imágenes que se ajusten al estándar 4:3 (como por ejemplo 640X480, 800X600, 1024X768...). Las que no cumplan esta relación de aspecto de 4:3 se transformarán para ocupar toda la ventana.

Figura 15: Ventana Imagen

Para el renderizado del modelo, se usa la librería PCL (véase Figura 16). La librería carga los vértices y polígonos que constituyen el modelo. Por defecto se aplica solid rendering. Para mejorar la percepción general del volumen del modelo se aplica iluminación global al renderizar. En el wired rendering al no tener superficie, también es más complicado apreciar volumen y por consiguiente más difícil la selección de vértices en el modelo.

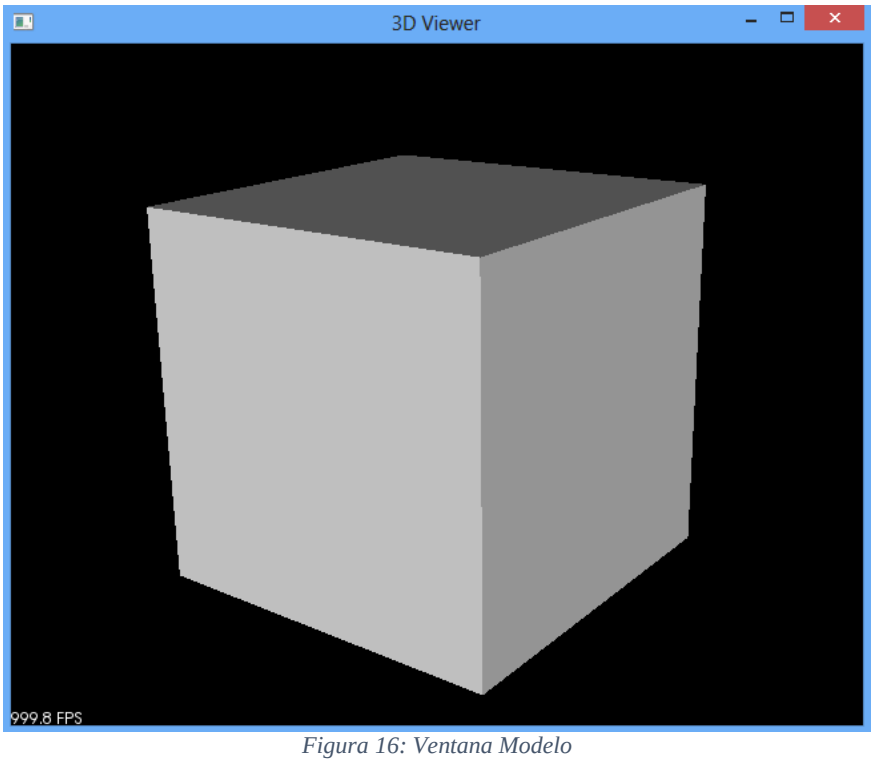


Figura 16: Ventana Modelo

Una vez se encuentran abiertas las dos ventanas con la imagen y el modelo, se procede al emparejamiento de puntos (véase Figura 17).

Para el emparejamiento se precisa seleccionar en la imagen un punto, que pueda tener correspondencia con algún vértice del modelo. La selección se realiza con el ratón mientras se presiona la tecla "shift". Posteriormente se realiza el mismo proceso en la ventana del modelo. Para realizar la selección se puede girar el modelo para obtener la proyección más adecuada que permita la selección de los vértices. Para seleccionar un vértice, se selecciona con el ratón mientras se presiona la tecla "shift".

La rotación del modelo se realiza seleccionando un punto cualquiera del modelo y arrastrando el ratón hacia el borde de la ventana mientras se mantiene seleccionado el punto. Dicha rotación se aplica tomando como referencia el origen de coordenadas del modelo.

El proceso descrito puede ser seguido a la inversa, seleccionado primero el vértice del modelo y luego el correspondiente punto de la imagen.

Los puntos seleccionados en la imagen y los vértices seleccionados en el modelo son almacenados internamente en la herramienta. Posteriormente pueden ser exportados a un fichero TXT. Dichos pares de puntos constituyen el input para el cálculo de las tareas siguientes.

Dada la dificultad de emparejar puntos de la imagen con puntos de la superficie del modelo, se fuerza a que sobre el modelo sólo se seleccionen vértices. La propia herramienta asignará la selección del punto al vértice más cercano a donde se haya hecho la selección con el ratón. De esta forma las asociaciones con los vértices del modelo cargado son más precisas y el error se produce en la selección de los puntos en la imagen.

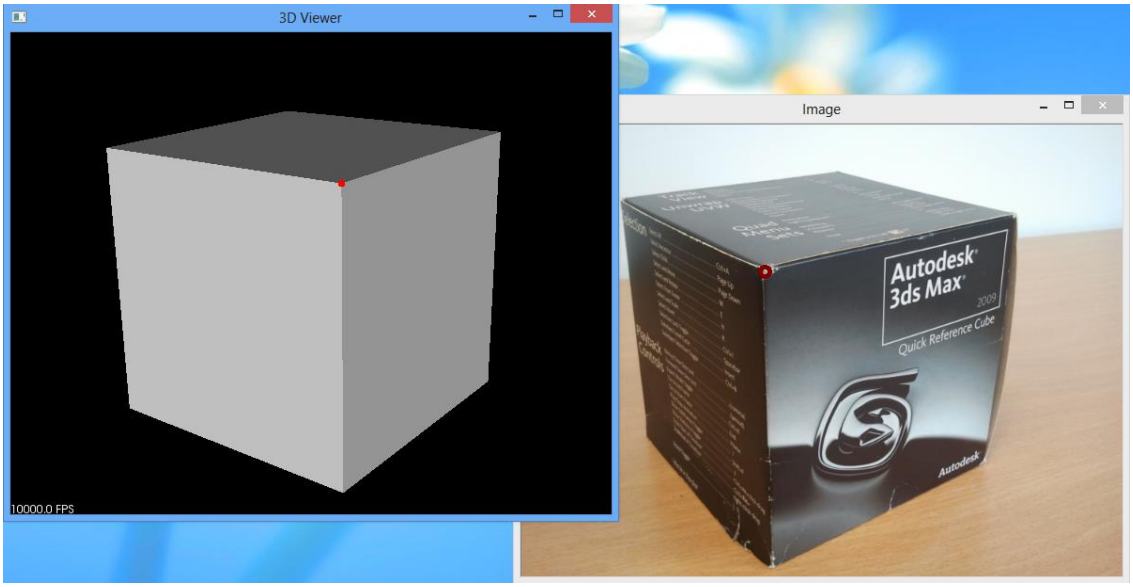


Figura 17: Emparejamiento de puntos

La selección de los pares de puntos debe ser como mínimo de cuatro pares de puntos. Es recomendable, si se puede, realizar más de cuatro emparejamientos para disminuir el error en las siguientes fases de la herramienta.

En caso de no seleccionarse un mínimo de cuatro pares de puntos, la ejecución terminará de forma prematura tras cerrar ambas ventanas de selección de puntos, y será necesario comenzar la ejecución de la herramienta desde el principio (véase Figura 18).

```
Puntos emparejados (1) :

Error!! Select at least 4 points --> Execution Aborted!!
```

Figura 18: Error: no se ha seleccionado el mínimo de cuatro puntos

Por otro lado existe una gestión mínima de errores en la selección de puntos, que asegura que por cada punto seleccionado en alguna de las dos ventanas, se selecciona su punto a emparejar en la ventana opuesta. Si surgiese la necesidad de anular la selección de un punto en cualquiera de las ventanas, esto no será posible (en la actual versión de la herramienta) y será necesario empezar de nuevo.

La ejecución de la herramienta queda pausada a la espera de la selección de más pares de puntos hasta que se cierren ambas ventanas de selección de puntos, tras lo cual se continuará la ejecución de los siguientes pasos de la herramienta. Esto se consigue mediante la sincronización del cierre de ambas ventanas en la ejecución principal.

3.2 Cálculo de los parámetros de una imagen y proyección de vértices del modelo en esa imagen

Terminología

- **Parámetros intrínsecos** (de una cámara): Son aquellos que definen la geometría interna y la óptica de la cámara. Son constantes en tanto no varíen las características y posiciones relativas entre la óptica y el sensor imagen. Un ejemplo donde varían estos parámetros, es cuando se aplica zoom en una cámara.
- **Parámetros extrínsecos** (de una cámara): Son aquellos que relacionan los sistemas de referencia del mundo real y la cámara describiendo la posición y orientación de la cámara en el sistema de coordenadas del mundo.

Una vez obtenidos los cuatro pares de puntos mínimos (imagen y modelo), todavía no es posible resolver el problema PNP (Perspective N Point) para la determinar la posición de la cámara con la que se tomó la imagen. Necesitamos para ello una referencia que será el sistema de referencia del mundo, en este caso será el sistema de referencia del modelo.

Para trabajar correctamente es necesario calibrar los parámetros intrínsecos de la cámara, que definen la geometría interna y la óptica de la cámara, o en el caso de disponer de ellos cargarlos. Estos parámetros son constantes en tanto no varíen las características y posiciones relativas entre la óptica y el sensor imagen.

Definimos el sistema de referencia, de la cámara (véase Figura 19). Para ello definimos su centro óptico C. En ese punto se coloca el sistema de referencia de la cámara. El eje Z será el *principal axis* de la cámara. Este eje es hacia donde mira la cámara. El eje Y será el vertical de la cámara.

Conocida la distancia focal f de la cámara, el punto (principal) P se encuentra en el eje Z^+ a una distancia f de C. Por P pasa un plano perpendicular a Z que será el *image plane*.

En este plano se define un sistema de referencia (sistema proyectado 2D) con su origen en P. Los ejes x, y son paralelos a los respectivos de la cámara.

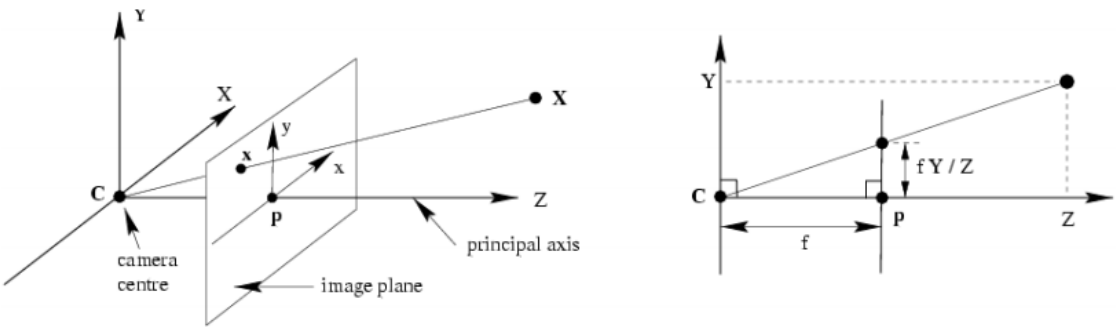


Figura 19: Distancia Focal

Según lo expuesto en la Figura 19, el paso de coordenadas del sistema de modelado al *image plane*, se modela, en coordenadas homogéneas, de la siguiente forma:

$$(X, Y, Z, 1) \mapsto ((f_x * X)/Z, (f_y * Y)/Z, 1)$$

Si lo desarrollamos:

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} f_x * X \\ f_y * Y \\ Z \end{pmatrix} = \begin{bmatrix} f_x & 0 & 0 & 0 \\ 0 & f_y & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} * \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} f_x * X \\ f_y * Y \\ Z \end{pmatrix} = \begin{bmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} * \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

De esta forma la matriz de proyección Pqueda definida como:

$$P = diag(f_x, f_y, 1)[I|0]$$

Por tanto los puntos proyectados en el *image plane* se modelan como x = PX, siendo X las coordenadas en el sistema de referencia de modelado y P la matriz de proyección que contiene los valores f_xy f_y de la cámara, tras haber sido obtenidos mediante algún proceso de calibración.

Se define otro sistema de referencia apropiado para trabajar con imágenes (sistema imagen), cuyo origen se sitúa en $(-p_x, -p_y)$ del sistema de proyección y con los ejes paralelos (véase Figura 20). Los valores p_x y p_y son las coordenadas del píxel central, que normalmente coincide con el centro del *image plane*.

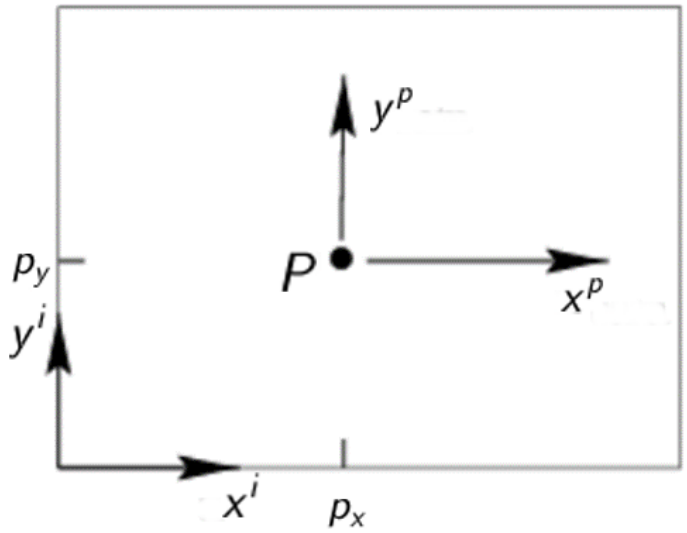


Figura 20: Punto Principal

La transformación será:

$$(X, Y, Z, 1) \mapsto (((f_x * X)/Z) + p_x, ((f_y * Y)/Z) + p_y, 1)$$

Si lo desarrollamos:

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} f_x * X + p_x \\ f_y * Y + p_y \\ Z \end{pmatrix} = \begin{bmatrix} f_x & 0 & p_x & 0 \\ 0 & f_y & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} * \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} f_x * X + p_x \\ f_y * Y + p_y \\ Z \end{pmatrix} = \begin{bmatrix} f_x & 0 & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} * \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

Se denomina matriz de calibración K a:

$$K = \begin{bmatrix} f_x & 0 & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix}$$

La matriz de calibración contiene la información referente a los parámetros intrínsecos de la cámara. Por tanto la matriz de proyección queda definida hasta el momento de la forma:

$$P = K[I|0]$$

Sea la matriz de transformación de coordenadas del sistema de referencia local del modelo al sistema de referencia de la cámara:

$$[R|t]$$

Entonces se cumplirá que la transformación del sistema local al de imagen quedará:

$$P = K[R|t]$$

Donde K es la matriz de calibración, R la matriz de rotación y t el vector de traslación.

Una vez que se dispone de la matriz de calibración y de los pares de puntos, se resuelve el problema PNP para conseguir los valores de la matriz de rotación y el vector de traslación. Para ello se utiliza la función *solvePnP* incluida en la librería OpenCV.

Una vez conocida la posición de la cámara (parámetros extrínsecos) se pueden proyectar los vértices del modelo en el sistema de referencia de la imagen. La proyección del modelo sobre la imagen se realiza usando la función *projectPoints* de la librería OpenCV. Los parámetros extrínsecos nos permiten posicionar y orientar el modelo pasando del sistema de referencia del modelo al de la cámara. La matriz de calibración

nos permite proyectar el modelo. De este modo se proyectan los vértices del modelo sobre la imagen (véase Figura 21).

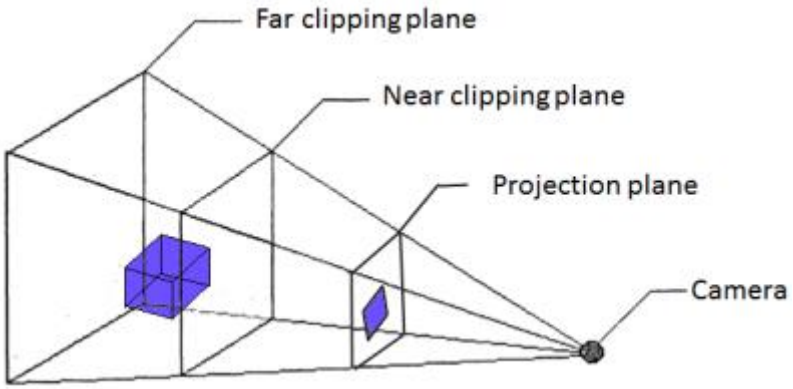


Figura 21: Plano de proyección

La proyección de los vértices del modelo debe dar como resultado la superposición de los vértices proyectados sobre los vértices del objeto en la imagen. Si existiera demasiado error de proyección de los puntos, éste puede minimizarse emparejando más puntos en la fase de emparejamiento de puntos descrito en la *Tarea 1* y recurriendo a algoritmos iterativos basados en la optimización como el de Levenberg-Marquardt.

3.3 Cálculo de la envolvente del modelo proyectado

Terminología

- **Matriz de adyacencia**: Es una matriz regular de orden NxN, donde cada posición de la matriz define si existe o no arista entre el vértice correspondiente a la fila y el vértice correspondiente a la columna.

Para el cálculo de la envolvente del modelo proyectado se genera una estructura de almacenamiento interno para mejorar el rendimiento, acceso y manipulación de los datos del modelo.

La estructura de almacenamiento interna utilizada se denomina GeometryOBJ y consta de:

- Vector de vértices (*points*)
- Array de aristas (*edges*)
- Vector de polígonos (*faces*)

El vector de vértices es una estructura que contiene coordenadas 3D usando la estructura *Point3f* de OpenCV. Dicha clase permite guardar las tres coordenadas (x, y, z) con precisión float. Estas coordenadas están expresadas en el sistema de referencia local del modelo.

El array de aristas es una estructura que se comporta como una matriz (estructura pseudo matricial). La estructura se accede como una matriz, pero la estructura usada (array) es una estructura de una única dimensión. Este acceso se realiza mediante una forma de acceso modular.

Sea la siguiente matriz de 3X3:

$$\begin{pmatrix} 5 & 3 & 7 \\ 4 & 1 & 4 \\ 8 & 5 & 9 \end{pmatrix}$$

El paso de una matriz a una estructura de una sola dimensión (pseudo matriz), se consigue encolando todas las filas y guardando el paso de la matriz que es el número de columnas (en este caso 3).

[5,3,7; 4,1,4; 8,5,9]

Si la numeración de las filas y las columnas de la matriz la iniciamos en el cero tendremos:

- De 0 hasta 2 filas
- De 0 hasta 2 columnas

Teniéndolo en cuenta, para acceder a la fila 1 y la columna 1, que contiene el valor 1, debemos aplicar la siguiente fórmula de acceso para saber a qué posición acceder en la estructura lineal:

$$\text{fila} * \text{paso} + \text{columna}$$

El resultado que nos devuelve es 4. Asumiendo que la numeración en la estructura lineal se inicia en el 0, la posición 4 corresponde al número 1.

La pseudo matriz de NxN de tamaño, toma como valor N el número de vértices que hay en el modelo. La información que se guarda internamente para cada posición sólo puede ser:

- 0 si no existe arista entre el vértice de la fila y la columna
- 1 si existe arista entre el vértice de la fila y la columna

Esta codificación establece que si el vértice de la fila i está conectado por una arista al vértice de la columna j, el vértice de la fila j también está conectado al vértice de la columna i por una arista. Esto es lo que se conoce como una matriz de adyacencia.

La diagonal principal de esta matriz de adyacencia puede rellenarse con 0 o 1, ya que por definición cualquier vértice es adyacente consigo mismo y no resulta de mayor interés su acceso.

Ejemplo de una matriz de adyacencia de 5 vértices:

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Como podemos ver la matriz resultante es simétrica y define para cada par de vértices, la existencia o no de arista. La matriz puede representarse gráficamente en forma de grafo 2D (véase Figura 22)

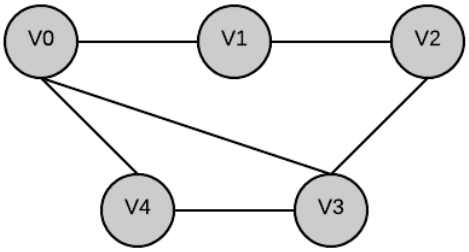


Figura 22: Grafo de Matriz de Adyacencia

El tercer y último elemento contenido en la estructura GeometryOBJ, el vector de polígonos, contiene la descripción de los polígonos del modelo usando la estructura face compuesta por:

- Tamaño del polígono (*fsize*)
- Array de referencias a los vértices que componen el polígono (*face*)

Nota aclarativa: "faces" traducido como "caras", es otra forma de denotar a los polígonos que delimitan la superficie.

Normalmente los polígonos suelen ser triángulos, aunque pueden ser polígonos de más lados. Por ello, para cada polígono se guarda el número de vértices que tiene.

Los índices que describen los vértices del polígono, indican la posición que ocupa cada vértice dentro del vector de vértices (*points*), siendo 0 el primer vértice.

Como beneficio, el uso de la estructura GeometryOBJ permite seguir trabajando internamente de la misma forma con cualquier modelo, sin importar el tipo de archivo desde el que se carga el modelo. Bastará con adaptar o crear un *parser* para el tipo de archivo que queramos cargar en la estructura.

Una vez disponemos de la estructura GeometryOBJ cargada con los datos del modelo y conociendo los vértices proyectados del modelo. Se puede dibujar la envolvente del modelo proyectado en la imagen usando la función *projectObject*.

La función *projectObject* de la clase IMGLoader permite varias formas de funcionamiento dependiendo su parametrización de los valores:

- Rellenado (fill)
- Sólo seleccionados (onlySlected)

El parámetro "rellenado" permite indicar a la función mediante un valor true o false si deseamos rellenar los polígonos que delimitan la superficie del modelo, o si por el contrario sólo queremos proyectar las aristas del modelo. Por otro lado, el parámetro "sólo seleccionados" permite proyectar y trabajar solamente con los vértices que hemos emparejado en el modelo durante la tarea de emparejamiento o con todos los vértices del modelo. Esta opción se indica rellenando el parámetro con un valor true o false.

El color utilizado para el dibujado de las aristas, así como para el rellenado de los polígonos, es el amarillo puro, (255, 255, 0) RGB.

Indicando que se quieren proyectar todos los vértices del modelo y que se quiere rellenar los polígonos de un color homogéneo (*fill=true & onlySelected=false*), se genera una imagen con la envolvente del modelo proyectado en la imagen original (véase Figura 23). Este output constituye el input de la siguiente tarea.

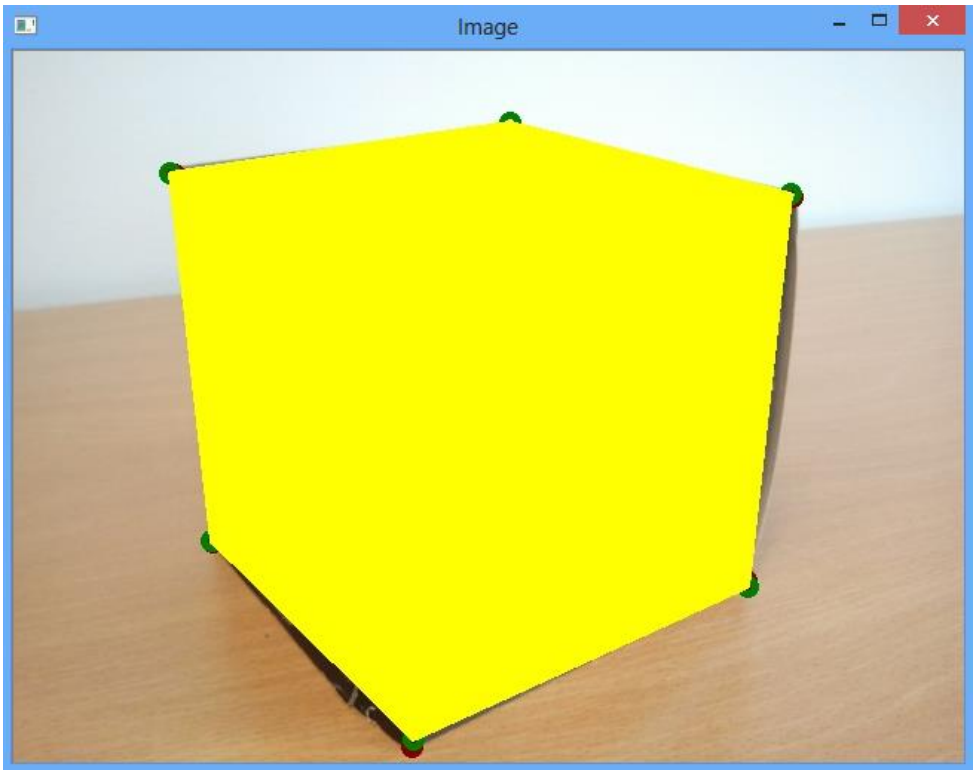


Figura 23: Envolvente del modelo

3.4 Generación de máscara de delimitación del modelo proyectado

La envolvente del modelo proyectado, generada en la tarea anterior nos permite localizar dónde se encuentra el objeto en la imagen. Al asignar un color homogéneo a la envolvente, resulta más fácil detectar dónde se encuentra el objeto en la imagen buscando dicho color. Sin la envolvente, las técnicas a utilizar serían más complejas.

Como primer paso se realiza una conversión del *espacio de color* RGB al espacio de color HSV mediante la función *cvCvtColor* de la librería OpenCV. Esto permite la detección y selección de objetos en la imagen, de manera un poco más simple, basada en los colores (véase Figura 24).

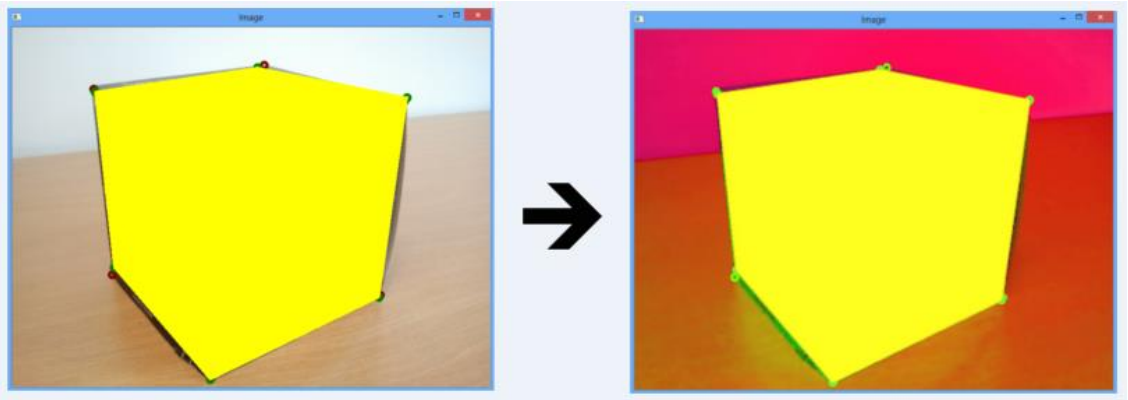


Figura 24: Conversión de RGB a HSV

El siguiente paso para la generación de la máscara, es seleccionar todos los píxeles que pueden ser parte del objeto. Se hará basándose puramente en sus valores HSV. OpenCV proporciona la función *cvInRangeS* que se puede utilizar para seleccionar píxeles basándose en su rango de valor cromático. Esto genera una máscara (véase Figura 25), una imagen binaria donde los píxeles de color blanco estaban dentro del rango especificado.

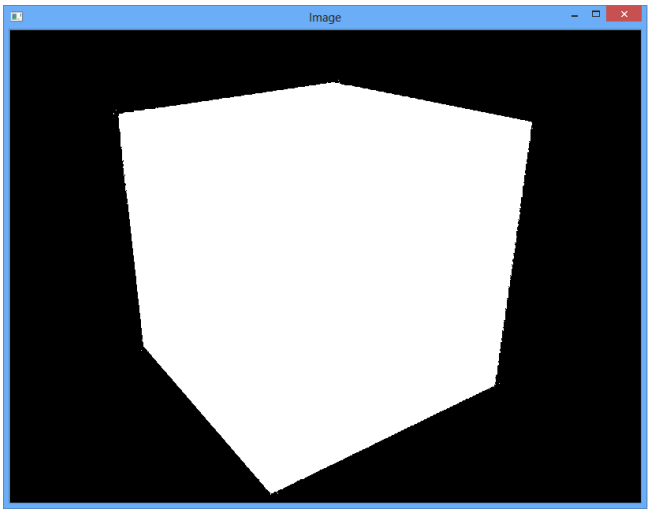


Figura 25: Máscara del objeto

Al haberse especificado manualmente el color amarillo para la envolvente del modelo proyectado, la asignación de los rangos para la creación de la máscara resulta simple. Únicamente es necesario definir un rango de color no muy amplio para el matiz y afinar un poco los valores de saturación y el del valor del color en la sección triangular del cono (véase Figura 26). De esta forma se obtiene un resultado correcto, sin necesidad de usar algoritmos de aprendizaje para la definición de los rangos automáticamente.

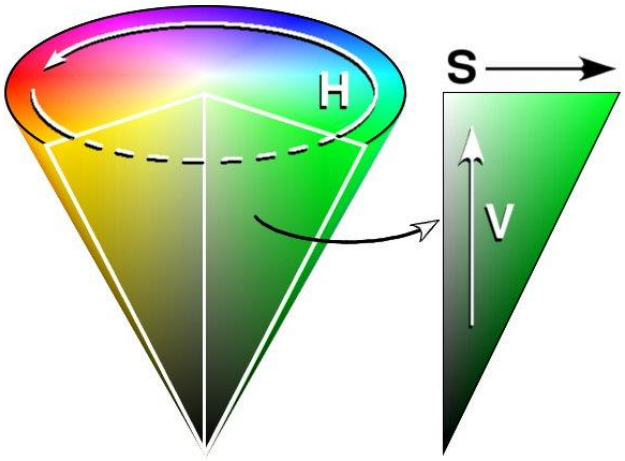


Figura 26: HSV

La principal ventaja de trabajar con HSV es que la selección del color se realiza en el primer parámetro (matiz). Así es más fácil lograr distintos tonos de un color específico. Esto da una ventaja frente a RGB. En RGB la parametrización del color hace difícil obtener distintos tonos de un color dado. Su naturaleza aditiva obliga a la asignación de un valor a cada uno de sus tres parámetros (Red, Green, Blue) para definir un color.

Una vez obtenida la máscara, la librería OpenCV proporciona métodos como *cvCopy* que permiten copiar imágenes aplicando máscaras. El uso de una máscara en la función, permite que sean copiados a la imagen de destino, aquellos píxeles de la sección blanca correspondiente al objeto. El resto de píxeles de la imagen son rellenados con un color homogéneo. De esta forma, se obtiene una imagen en la cual, sólo aparece el objeto que se pretende aislar de la imagen (véase Figura 27).

La imagen con el objeto aislado, constituye el input de la siguiente tarea.

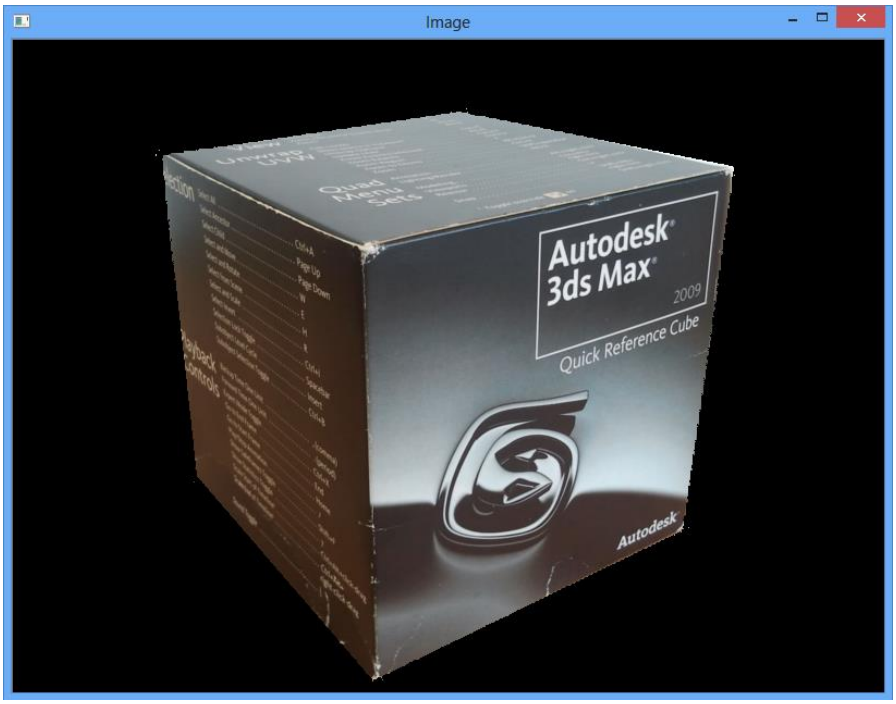


Figura 27: Objeto aislado de la imagen

3.5 Obtención de *Keypoints* y Descriptores del objeto

Terminología

- **Grid**: Estructura que divide un área en regiones. Un ejemplo habitual es la cuadrícula para presentar datos en forma de tabla.
- **Movimientos sacádicos**: Son cada uno de los movimientos y fijaciones que realiza el ojo para que la imagen caiga sobre el centro de la retina que es la más precisa. Razón por la cual nuestra imagen es nítida en el centro y borrosa en la periferia. Por lo tanto el ojo al sólo disponer de una mínima zona de visión precisa y nítida, realiza esos movimientos para registrar el mayor número de elementos posibles del entorno.

Disponiendo de la imagen del objeto, aislado de su entorno en la tarea anterior, hay que obtener los *keypoints* del objeto y los descriptores que nos permitan identificarlos en otras imágenes.

Los *keypoints*, o también denominados puntos de interés, son puntos que son lo suficientemente significativos en la apariencia del objeto como para poder ser identificados de nuevo. Para la obtención de los *keypoints* se usará el algoritmo FAST [RD06] ofrecido dentro de la librería OpenCV.

En la bibliografía existen distintos criterios para definir descriptores y determinar *keypoints*. Para la detección de *keypoints* es frecuente trabajar en escala de grises, es decir, para trabajar con la intensidad de brillo de cada pixel. Esto mejora el rendimiento y la capacidad de detección de los algoritmos.

El algoritmo FAST opera considerando una circunferencia de dieciséis píxeles alrededor del píxel candidato a *keypoint* I_p (véase Figura 28). El detector FAST clasificará p como keypoint si existe un conjunto de N píxeles en la circunferencia que sean todos más brillantes que la intensidad de píxel candidato I_P más el umbral t. El pixel ptambién es clasificado como *keypoint* si el conjunto de N píxeles en la circunferencia son más oscuros que $I_p - t$.

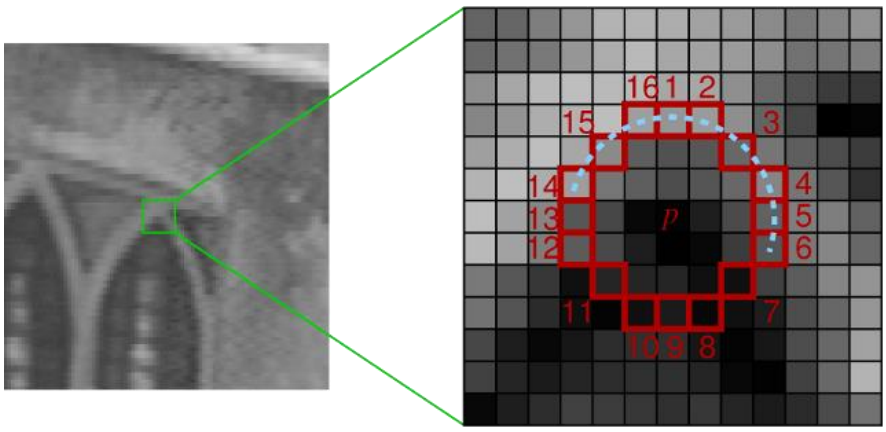


Figura 28: Detector de Esquinas FAST

La función *compute* de la clase *FAST* de la librería OpenCV, nos permite aplicar el algoritmo a la imagen del objeto. Los parámetros que definen el comportamiento del algoritmo son el *threshold* y *nonmaxSupression*. El primero define el umbral (explicado en el párrafo anterior) mediante un valor entero. El segundo define si se aplica la supresión de los puntos no máximos adyacentes. Esto resuelve el problema de que una esquina quede sobredeterminada por *keypoints* correspondientes a píxeles adyacentes (véase Figura 29).

La supresión de los puntos no máximo adyacentes, se realiza para los píxeles próximos entre sí que son candidatos a *keypoint*. Para quedarse con el *keypoint* más representativo de la zona, se realiza para cada *keypoint* el sumatorio de las diferencias absolutas entre p y sus dieciséis píxeles circundantes. El *keypoint* que consiga el valor más alto se quedará como keypoint y los restantes más próximos no serán *keypoints*.

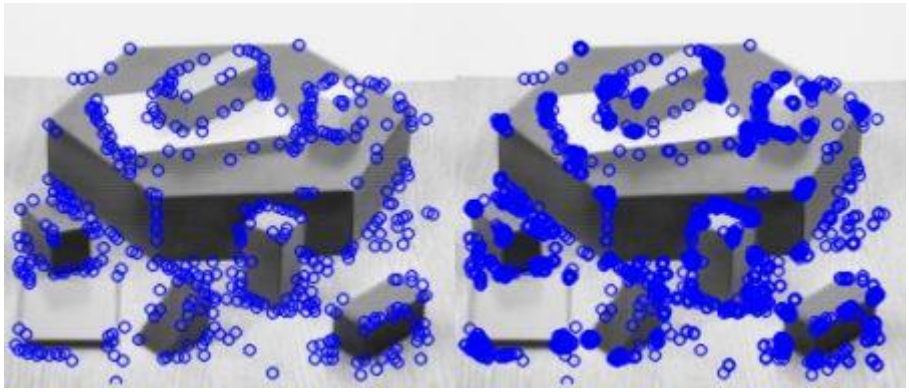


Figura 29: FAST nonmaxSuppression

En la Figura 29 se muestra la diferencia entre la aplicación o no de la supresión de los no máximos adyacentes. En la parte de la izquierda se aplica la supresión y en la derecha no se aplica la supresión.

En el caso de la herramienta no nos interesa tener sobredeterminada ninguna región. Resulta óptimo tener los *keypoints* repartidos lo más homogéneamente posible, permitiendo tener un conocimiento más completo y homogéneo de los puntos de interés del objeto.

Durante la ejecución de la herramienta no se muestra la primera extracción de *keypoints*, pero sí los mejores *keypoints*. La Figura 30 muestra la primera extracción de *keypoints* con el algoritmo FAST.

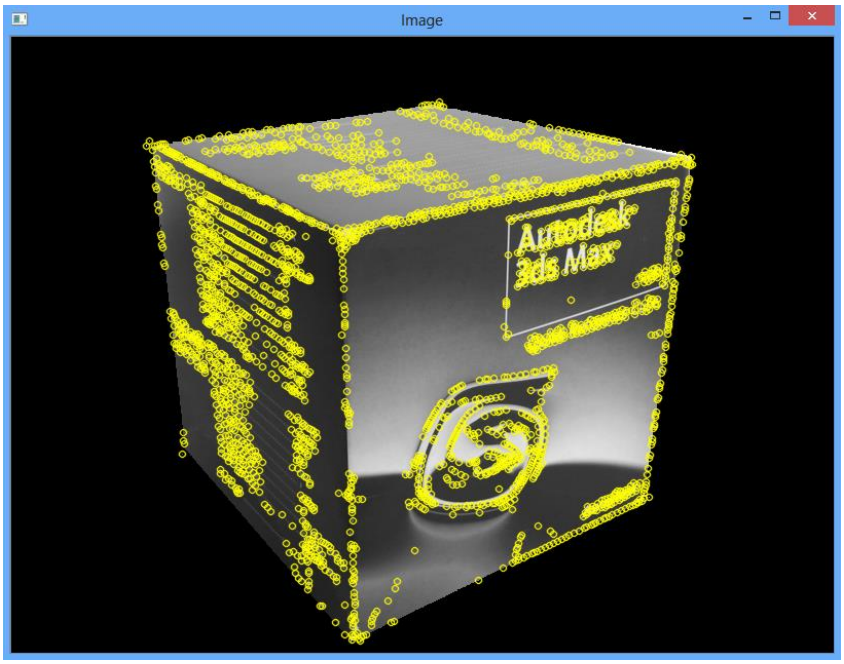


Figura 30: Resultado de aplicar FAST

En ocasiones la delimitación del objeto en la imagen no es del todo precisa y puede dejar píxeles sueltos (véanse en la Figura 31 los dos píxeles sueltos junto a la esquina). Estos píxeles son *keypoints* espurios ya que no corresponden a la envolvente del modelo y serán filtrados en la siguiente tarea (*Unprojection*).

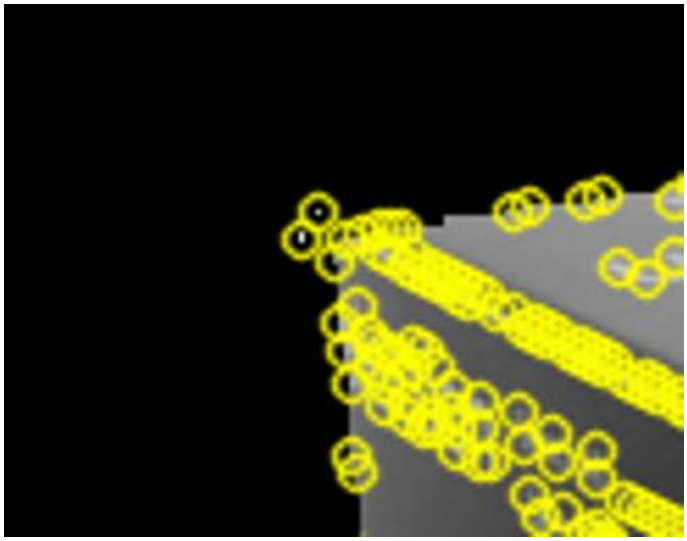


Figura 31: Keypoints espurios en la esquina

Para determinar el valor de los descriptores usaremos FREAK. El descriptor FREAK está inspirado en el sistema de visión humano, más concretamente en la retina. Debido a esto, el descriptor se llamó “Fast Retina Keypoint”, abreviado como FREAK.

El descriptor permite la comparación binaria de manera eficiente, de pares de intensidades de los píxeles de la imagen, sobre un patrón de muestreo retina. La selección de pares de intensidades, produce un patrón altamente estructurado que imita a la búsqueda mediante *movimientos sacádicos* de los ojos humanos. El patrón de muestreo retina utiliza un *grid* de muestreo circular con una alta densidad de puntos cerca del centro (véase Figura 32). Este tema está desarrollado con mayor detalle en el trabajo de Alexandre Alahi [AOV12].

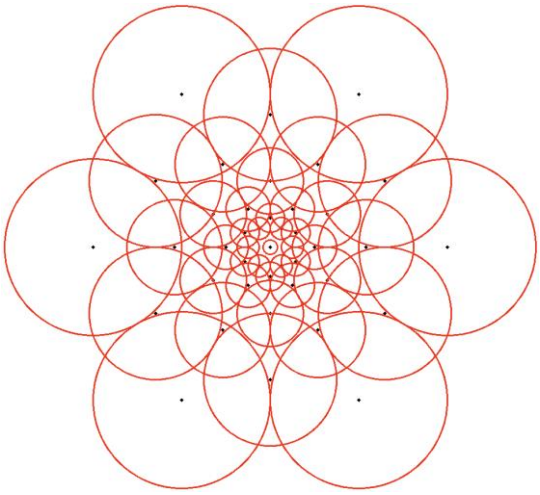


Figura 32: Grid de muestreo circular de FREAK

La función *compute* de la clase *FREAK* de la librería OpenCV, nos permite calcular los descriptores de los *keypoints*, utilizando el algoritmo FREAK.

3.6 *Unprojection*

Dados los *keypoints* de la imagen obtenidos en la tarea anterior, se van a determinar sus correspondientes puntos 3D en la superficie del modelo. A continuación se expone la resolución gráfica del problema.

El paso de puntos 2D a 3D constituye un sistema compatible indeterminado dependiente del valor de Z. Esto nos impide saber con precisión el valor concreto de Za partir de unas coordenadas x e y de la imagen (véase Figura 33).

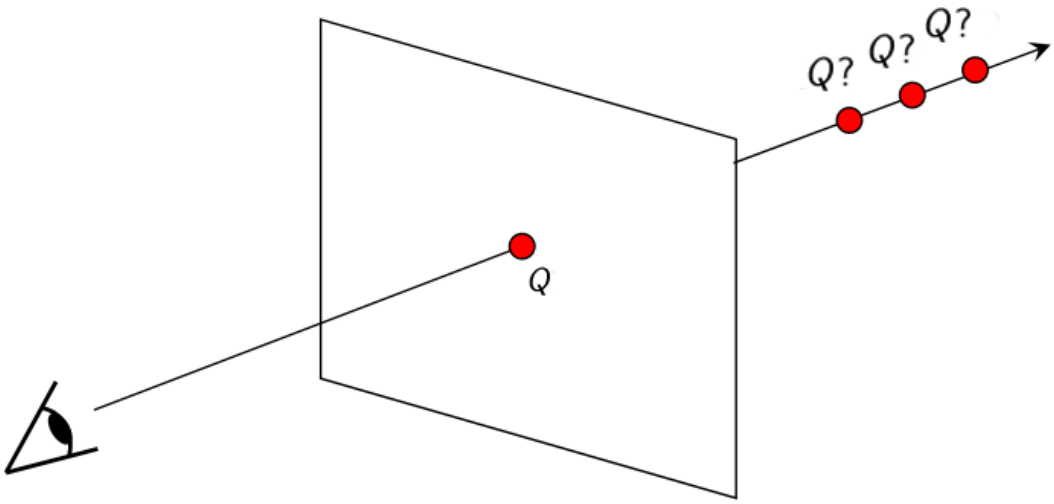


Figura 33: Dado un punto proyectado, infinitos puntos 3D han podido dar origen a esa proyección

Para el ojo humano también resulta un problema el resolver la relación entre elementos de una imagen cuando no se conoce a priori las dimensiones y las ubicaciones. Esto crea una duda razonable en la percepción del mundo. Claros ejemplos son los que se pueden apreciar en la Figura 34 y la Figura 35.

Figura 34: Perspectiva Calabaza

Figura 35: Perspectiva Torre de Pisa

Nuestro sistema de visión y concepción del mundo funciona con modelos conocidos que vamos aprendiendo a lo largo de nuestra vida. Nos permite establecer posiciones y dimensiones para los elementos que percibimos a nuestro alrededor. Aunque obviamente el cerebro también reconstruye y crea nuevos modelos que nos permiten entender la realidad a partir de la perspectiva, el sombreado o incluso el movimiento de un objeto.
En el caso de la herramienta, la forma de entender un espacio 3D y detectar objetos a partir de un sistema mono-vista (imágenes en 2D), es la utilización de modelos que representen objetos de la realidad.

Modelado matemáticamente el problema de paso de puntos 3D a 2D queda de la siguiente forma:

$$P_{2D} = K * P_{3D}$$

Donde K es la matriz de parámetros intrínsecos de la cámara. En este apartado el sistema de modelado y de la cámara coinciden.

Si desarrollamos la ecuación:

$$\begin{pmatrix} x_{2D} \\ y_{2D} \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X_{3D} \\ Y_{3D} \\ Z_{3D} \\ 1 \end{pmatrix}$$

Despejando las coordenadas 2D se obtiene:

$$x_{2D} = f_x * X_{3D} + c_x * Z_{3D} = (X_{3D}/Z_{3D}) * f_x + c_x$$

$$y_{2D} = f_y * Y_{3D} + c_y * Z_{3D} = (Y_{3D}/Z_{3D}) * f_y + c_y$$

Despejando X_{3D} y Y_{3D}, quedan en función de Z_{3D}.

$$X_{3D} = (x_{2D} - c_x) * Z_{3D}/f_x$$

$$Y_{3D} = (y_{2D} - c_y) * Z_{3D}/f_y$$

Para encontrar los puntos 3D del objeto se dibuja el objeto. Se usa la librería GLUT basada en OpenGL. Tras dibujar el objeto la librería permite acceder al z-buffer. Así, dado un *keypoint*, sabemos su coordenada z leyéndola del z-buffer. A partir de ella, obtenemos las otras dos coordenadas del *keypoint* en el sistema de referencia de la cámara

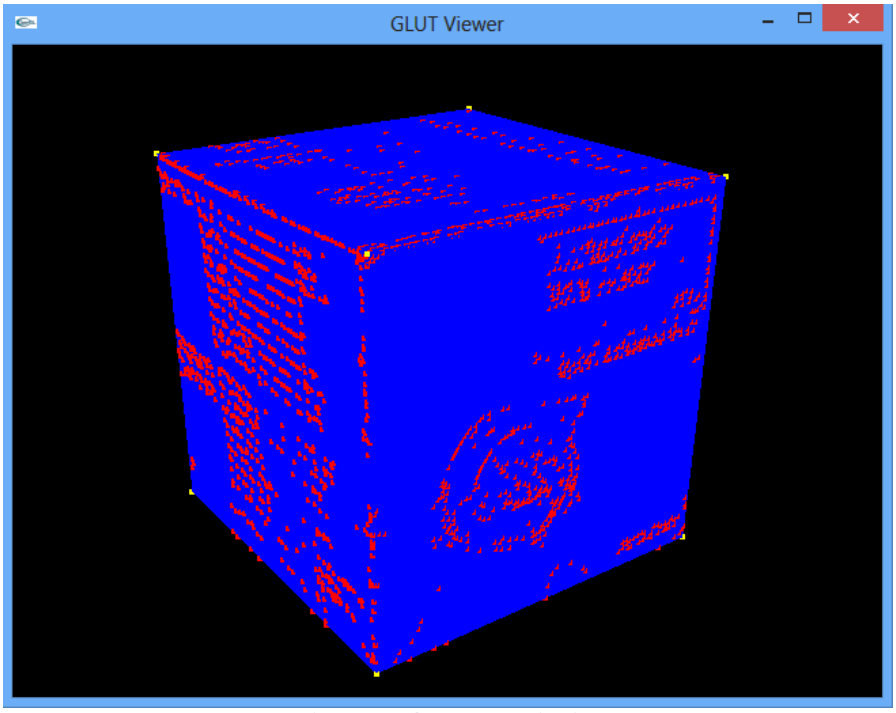


Figura 36: GLUT Unprojection

Si un valor leído del z-buffer está en el infinito (far plane), entonces el *keypoint* es espurio y no pertenece al objeto. Dicho punto es eliminado de la lista de *keypoints*, junto a sus correspondientes descriptores.

En la Figura 36 se pueden apreciar:

- Puntos amarillos correspondientes a los vértices del modelo
- Polígonos renderizados de color azul
- Puntos rojos correspondientes a los *keypoints* en el interior o frontera de los polígonos renderizados.

Los *keypoints* y descriptores filtrados correspondientes al objeto, junto con las coordenadas 3D de los *keypoints*, constituyen el input para las siguientes tareas.

3.7 Guardado del Modelo de *Tracking*

Terminología

- **3DTM**: Abreviatura de *3D Trackable Model*

Una vez obtenidos las coordenadas 3D de los *keypoints*, es posible generar un *3D trackable model* (*3DTM*) con la información obtenida hasta el momento. La información que define un *3DTM* se guarda en un fichero YAML.

La principal información que define un *3D trackable model* es:

- *Keypoints* (puntos de interés)
- *Descriptors* (descriptores)
- *3D Point* (coordenadas 3D de los *keypoints*)

Los *keypoints* son los puntos de interés obtenidos de la imagen de entrenamiento en la *Tarea 2*. No permiten una detección del objeto, pero contienen las coordenadas 2D que localizan cada punto de interés en la imagen.

Los *descriptors* contienen la información identificativa que describe la región donde se sitúa el *keypoint*. La información que contienen permite volver a localizar un *keypoint* en la imagen o en otra imagen que contenga el objeto.

Los 3D Point son las coordenadas 3D correspondientes a los *keypoints* calculados para el objeto mediante el procedimiento de *Unprojection* descrito en la *Tarea 6*.

Otra información de interés que se guarda es:

- Matriz de parámetros intrínsecos de la cámara.
- Nombre de la imagen de entrenamiento (incluida la extensión) usado para la extracción de los *keypoints*.
- Nombre del modelo (incluida la extensión) usado para la *Unprojection*.

El fichero YAML es guardado con el nombre "3DTM.yml". Para obtener la información completa de los distintos valores guardados en el fichero YAML del *3D trackable model*, véase el Anexo I.

Existe la posibilidad de guardar el modelo dentro del fichero YAML. Esto es posible, siempre y cuando todos los polígonos del modelo sean del mismo tipo y los tipos de polígonos sean triángulos o cuadrados. Resulta interesante esta opción cuando se quiere tener un único archivo serializado con toda la información necesaria para realizar el *tracking* de un objeto.

La elección del uso YAML como estándar de serialización de objetos viene condicionada por dos factores:

- La lectura y escritura de ficheros YAML viene incluida dentro de la librería

OpenCV
- Resulta legible para un ser humano y a la vez está estructurado y resulta eficiente

Una vez se han guardado todos los valores en el fichero YAML, la herramienta realiza un empaquetado en formato ZIP (véase Figura 37) con:

- Fichero YAML
- Imagen de entrenamiento
- Modelo

Para el empaquetado ZIP de la información se ha usado el código fuente de Info-Zip.

La herramienta genera el nombre del fichero siguiendo el patrón "3DTM(nombre_imagen).zip". En caso concreto de la Figura 37 el nombre queda como "3DTM(autodesk_cube).zip" porque el nombre de la imagen era "autodesk_cube.png".

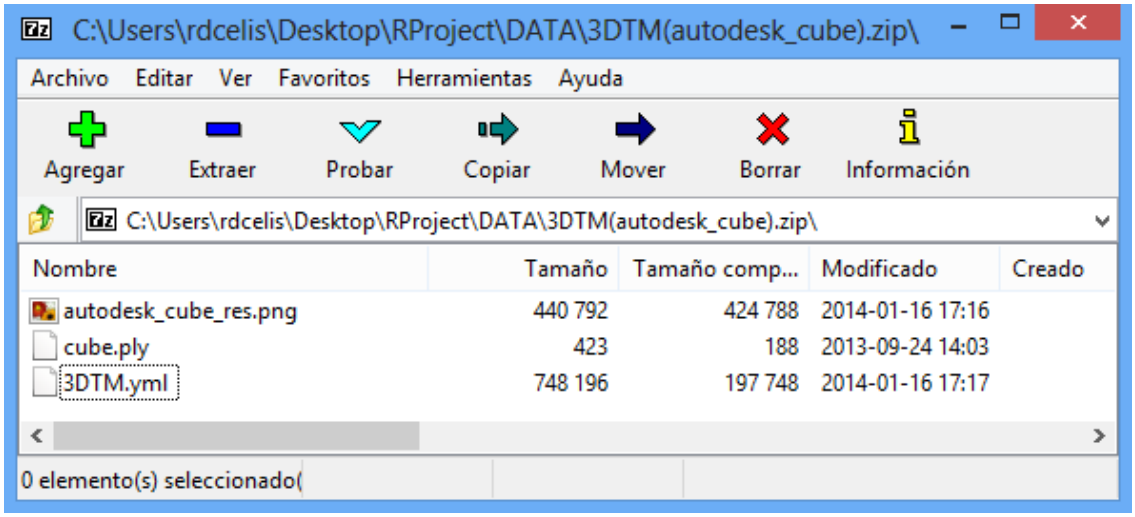


Figura 37: Contenido del fichero ZIP

3.8 Detección de *Keypoints* del Objeto en una nueva imagen

Dados el *3D trackable model* (3DTM) y una nueva imagen del objeto, se realiza la detección del objeto en la nueva imagen a partir del 3DTM. Esta tarea tiene un valor probatorio que verifica que a partir del 3DTM se puede volver a detectar de nuevo el objeto en una nueva imagen.

En primer lugar se obtienen los *keypoints* de la nueva imagen usando FAST y sus descriptores usando FREAK, tal y como se describe en la *Tarea 5*. Una vez se conocen los *keypoints* y los descriptores, se realiza un *matching* masivo entre los descriptores de la nueva imagen y los del 3DTM. Como ejemplo ilustrativo del *matching* masivo, se puede ver en la Figura 38 la etapa incipiente del *matching* con unos pocos emparejamientos. En la Figura 39 puede observarse el *matching* masivo completado.

Para ello se utiliza *BFMatcher*, un algoritmo de fuerza bruta disponible en la librería OpenCV. El algoritmo debe recibir para su funcionamiento la norma utilizada para definir la *distancia* entre cada par de descriptores. En el caso de la herramienta se ha usado *NORM_L2* que se define como la *distancia* euclídea:

$$NORM_{L2} = ||src1 - src2||_{L2} = \sqrt{\sum_I (src1(I) - src2(I))^2}$$

Para cada descriptor en el primer set, *BFMatcher* encuentra el descriptor más cercano en el segundo set, comparando cada uno de los descriptores del primer set con todos los descriptores del segundo set. De esta forma cada descriptor del primer set, queda emparejado con un único descriptor del segundo set.

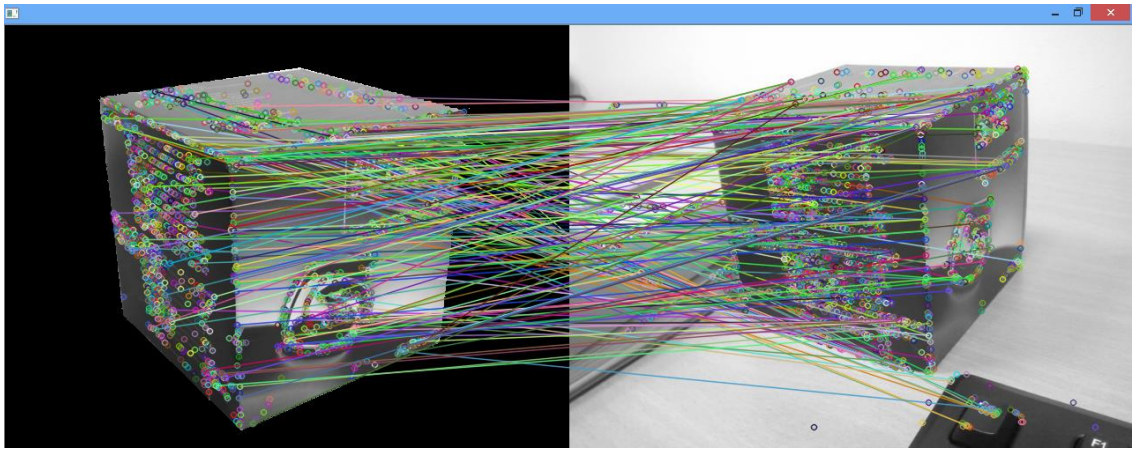


Figura 38: Matching masivo parcial

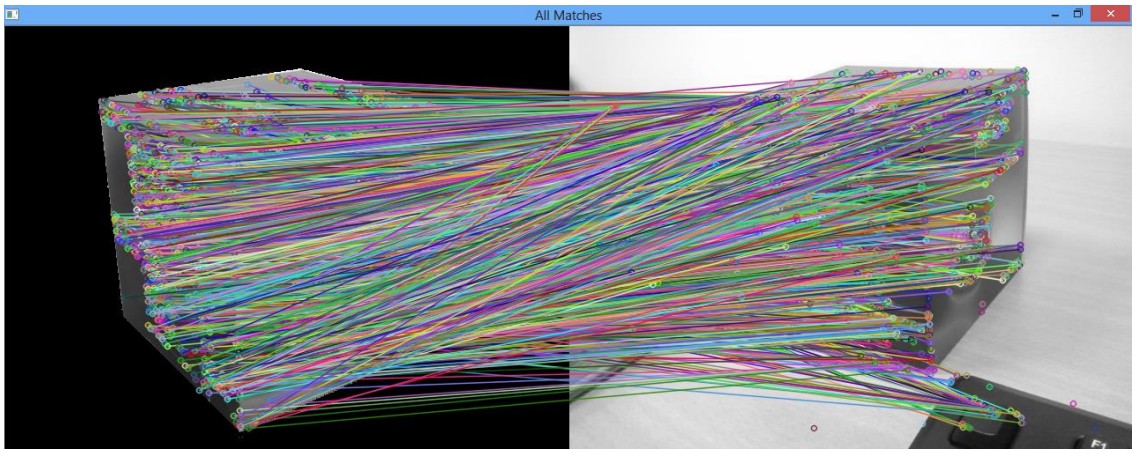


Figura 39: Matching masivo completo

Una vez se ha ejecutado el algoritmo, se filtran los mejores emparejamientos estableciendo un umbral máximo para la *distancia* que puede haber. Este valor puede ser asignado manualmente o entrenado para conseguir el mejor set de *keypoints*.

Tras filtrar los mejores emparejamientos del *matching* masivo, quedan los mejores *keypoints* (véanse Figura 30 para antes del filtrado y para después Figura 40).

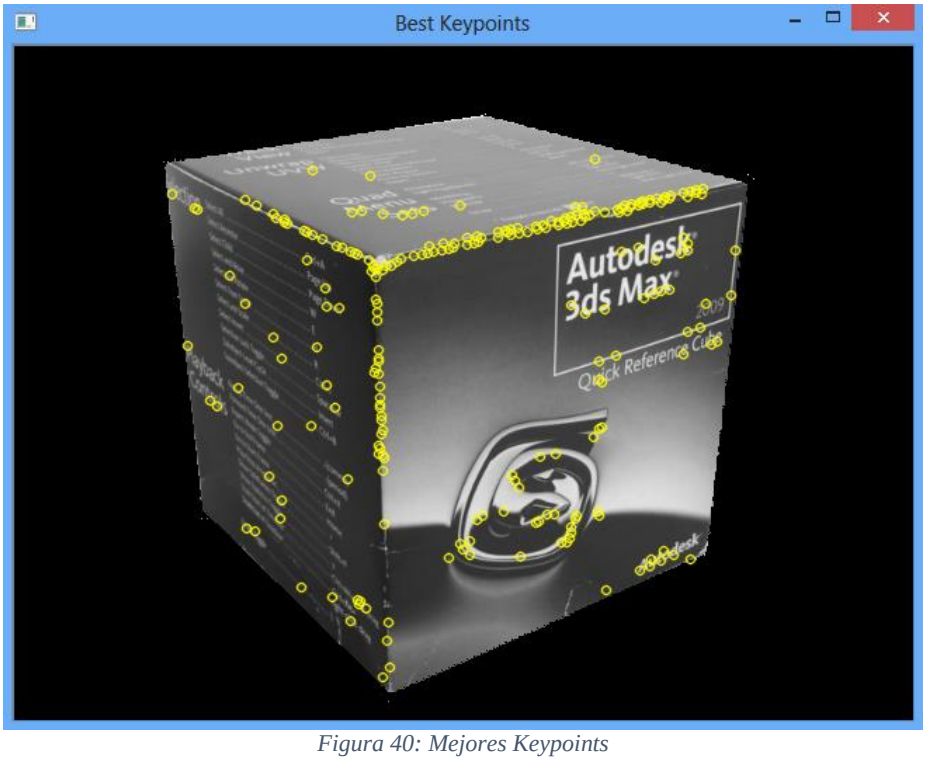


Figura 40: Mejores Keypoints

Se podrían probar otros algoritmos que bien produzcan mejores emparejamientos, o bien reduzcan los tiempos de cálculo.

Finalmente habría que hacer notar lo siguiente. Cuando el objeto varía su posición (se aleja o acerca) respecto a la posición inicial (imagen de entrenamiento), los descriptores cambian. Esto es debido a que los píxeles próximos a cada *keypoint* varían (véase Figura 41 y Figura 42). El uso de algoritmos multiescala (robustos frente al cambio de escala), puede solventar el problema de obtener demasiados emparejamientos erróneos con *distancias* mayores al umbral de *distancia*.

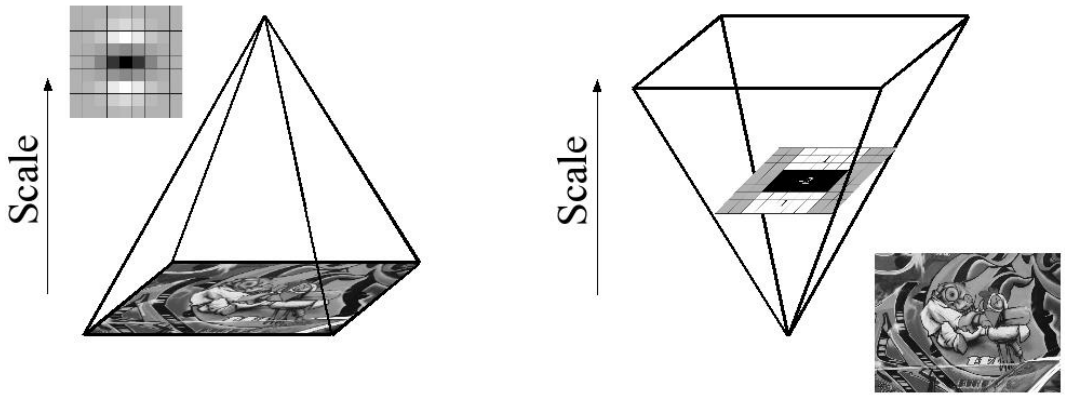


Figura 41: Análisis de escala

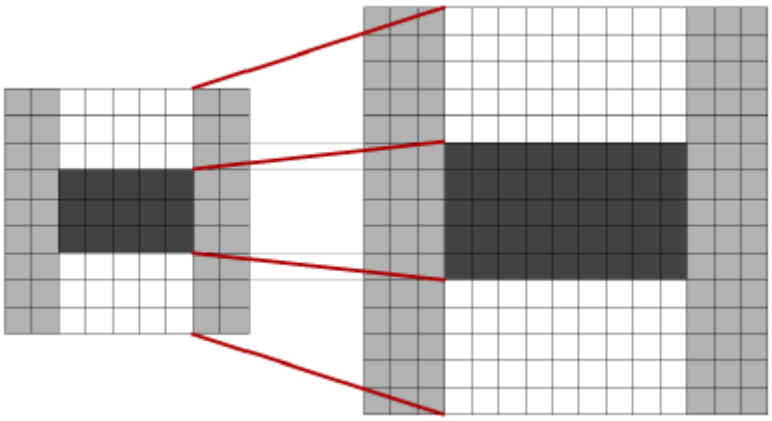


Figura 42: Normalización del espacio

Existen otros problemas a la hora de realizar una detección correcta como la cantidad de luz, el tipo de luz, el blur o el viewport, que igualmente están ligados a las características de los algoritmos usados durante el proceso de detección.

Los *keypoints* filtrados con sus correspondientes coordenadas 3D en el modelo y sus descriptores, constituyen el input de la siguiente tarea.

3.9 Cálculo iterativo de los parámetros de una imagen y proyección del modelo en esa imagen

Se dispone de un 3DTM y de una nueva imagen procesada en el paso anterior. Se dispone de los mejores *keypoints* de la nueva imagen y sus correspondientes coordenadas en el modelo.

En esta fase se calcularán los parámetros extrínsecos de la imagen (rotación y translación) para la nueva imagen. También se conocen los parámetros intrínsecos de la cámara (matriz de calibración) descritos en la *Tarea 2*.

Esta vez el cálculo de la matriz se resuelve de manera iterativa usando la función *solvePnPRansac* de la librería OpenCV. La función recibe como en el caso de *solvePnP* los pares de puntos (mejores *keypoints* y sus coordenadas del modelo) y la matriz de calibración. Además de estos parámetros se define el máximo de iteraciones, el mínimo de inliers y el error de re-proyección:

- El máximo de iteraciones es el número de veces que se ejecuta el algoritmo si no se llega al mínimo de inliers definido.

- El mínimo de inliers es el número de inliers que si son encontrados en alguna etapa del algoritmo, finaliza la ejecución del algoritmo.

- El error de re-proyección, es el valor máximo permitido para la *distancia* entre el punto observado y el punto proyectado, para ser considerado el punto como un inlier.

El algoritmo además de devolvernos la rotación y translación del modelo, devuelve un vector que contiene los índices de los inliers.

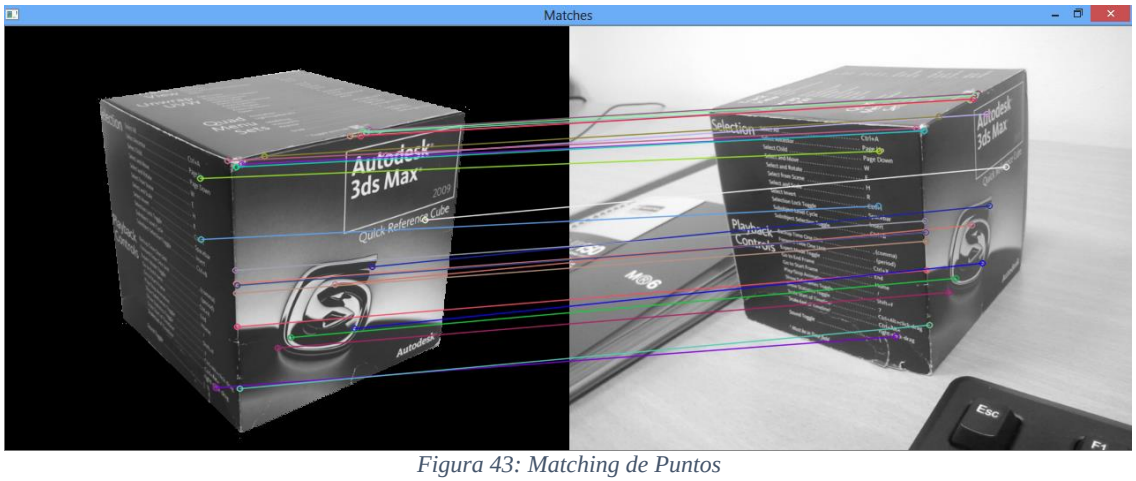


Figura 43: Matching de Puntos

Los inliers válidos son mostrados por pantalla tras la finalización del algoritmo. A la izquierda se muestra la imagen de entrenamiento sin su entorno y a la derecha la nueva imagen donde se ha detectado el objeto. En ellas se dibujan las correspondencias entre los *keypoints* del modelo 3DTM y los inliers de la nueva imagen. Este *matching* prueba que sigue habiendo una correspondencia entre el modelo 3DTM y los inliers detectados para la nueva imagen (véase Figura 43).

A continuación se proyecta el modelo sobre la imagen usando *projectObject* de la clase IMGLoader. Internamente la función usa *projectPoints* de la librería OpenCV para proyectar el modelo, permitiendo además dibujar las aristas y vértices del modelo, junto a los inliers del modelo proyectado (véase Figura 44).

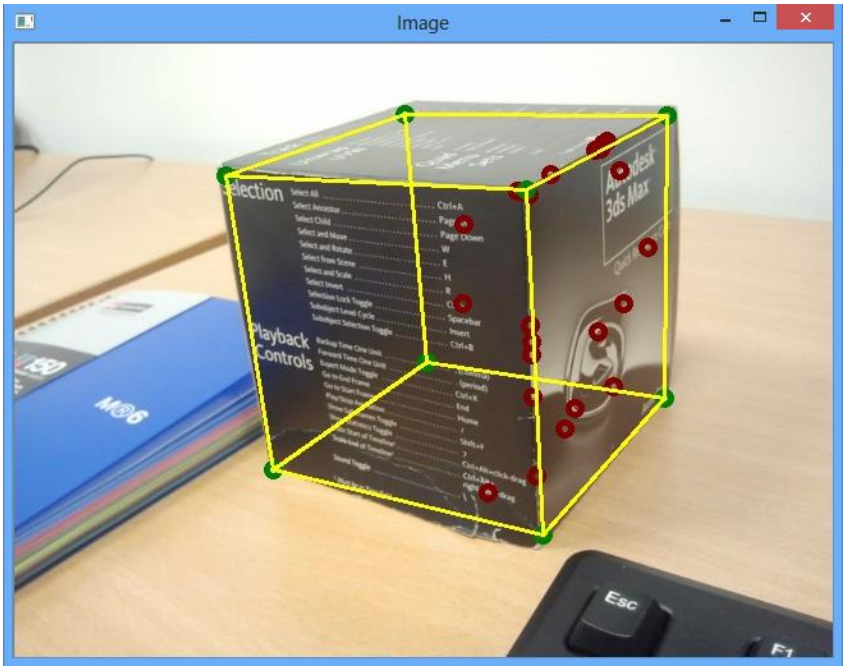


Figura 44: Proyección del modelo a partir de keypoints

En la Figura 44 las aristas del modelo están representadas por las líneas amarillas, los vértices del modelo por los círculos verdes y los inliers (*keypoints*) por círculos de color rojo.

Obsérvese que esta vez el proceso se realiza sin necesidad de intervención por parte del usuario (selección de puntos).

Finalmente conviene saber que se puede repetir la detección y proyección del objeto con cuantas nuevas imágenes se desee. Para ello se repetirán los procedimientos descritos en las *Tareas 8 y 9*.

Por otro lado con el modelo 3DTM y utilizando algoritmos de detección y *matching* más eficientes y optimizados, es posible ejecutar los procedimientos de las *Tareas 8 y 9* en tiempo real. Esto puede ser durante la grabación o reproducción de un vídeo. De esta forma será posible realizar un seguimiento de un objeto 3D con textura, a partir del 3DTM generado por la herramienta.

BIBLIOGRAFÍA

En este apartado se enumeran las distintas fuentes de información consultadas durante el proyecto. La clasificación se realiza en publicaciones y referencias Web y dentro de cada clasificación se ordenan las referencias alfabéticamente

Publicaciones

[A97] AZUMA, Ronald T., et al. A survey of augmented reality. *Presence*, 1997, vol. 6, no 4, p. 355-385.

[AOV12] ALAHI, Alexandre; ORTIZ, Raphael; VANDERGHEYNST, Pierre. Freak: Fast retina keypoint. En *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012. p. 510-517.

[BPG10] BARANDIARAN, Iñigo; PALOC, Céline; GRAÑA, Manuel. Real-time optical markerless tracking for augmented reality applications. *Journal of Real-Time Image Processing*, 2010, vol. 5, no 2, p. 129-138.

[BS13] BARBADILLO, Javier; SÁNCHEZ, Jairo R. A Web3D authoring tool for augmented reality mobile applications. En *Proceedings of the 18th International Conference on 3D Web Technology*. ACM, 2013. p. 206-206.

[BSG13] BARRENA, Nagore; SÁNCHEZ, Jairo R.; GARCÍA-ALONSO, Alejandro. A Distributed and Collaborative vSLAM Framework for Real-Time Localisation in Huge Environments for Mobile Devices. En *Eurographics 2013-Posters*. The Eurographics Association, 2013. p. 11-12.

[D00] DELLAERT, Frank, et al. Structure from motion without correspondence. En*Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*. IEEE, 2000. p. 557-564.

[JD02] JURIE, Frédéric; DHOME, Michel. Hyperplane approximation for template matching. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 2002, vol. 24, no 7, p. 996-1000.

[K05] KARLSSON, Niklas, et al. The vSLAM algorithm for robust localization and mapping. En *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. IEEE, 2005. p. 24-29.

[LF05] LEPETIT, Vincent; FUA, Pascal. *Monocular model-based 3d tracking of rigid objects: A survey*. Now Publishers Inc, 2005.

[MK94]MILGRAM, Paul; KISHINO, Fumio. A taxonomy of mixed reality visual displays.*IEICE TRANSACTIONS on Information and Systems*, 1994, vol. 77, no 12, p. 1321-1329.

[RD06] ROSTEN, Edward; DRUMMOND, Tom. Machine learning for high-speed corner detection. En *Computer Vision–ECCV 2006*. Springer Berlin Heidelberg, 2006. p. 430-443.

[VLF04] VACCHETTI, Luca; LEPETIT, Vincent; FUA, Pascal. Stable real-time 3d tracking using online and offline information. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 2004, vol. 26, no 10, p. 1385-1391.

Referencias Web (Enero 2014)

FAST - Corner Detection http://www.edwardrosten.com/work/fast.html

FREAK - Fast Retina Keypoint http://www.ivpe.com/freak.htm

OpenCV http://opencv.org/

- Manual de referencia http://docs.opencv.org/2.4.6/modules/refman.html
- Guía de Usuario http://docs.opencv.org/2.4.6/doc/user_guide/user_guide.html
- Tutoriales http://docs.opencv.org/2.4.6/doc/tutorials/tutorials.html

PCL http://pointclouds.org/

- Documentación API http://docs.pointclouds.org/1.6.0
- Tutoriales http://pointclouds.org/documentation/tutorials

Combinación de Extractores y Descriptores de puntos y regiones de interés (PFC)

http://arantxa.ii.uam.es/~jms/pfcsteleco/lecturas/20131022FulgencioNavarroFajardo.pdf

Stack Overflow http://stackoverflow.com

Microsoft MSDN http://msdn.microsoft.com

Zip Utils http://www.wischik.com/lu/programmer/zip_utils.html

Info-Zip http://infozip.sourceforge.net

Wikipedia http://www.wikipedia.org

Binarios de RProject (Windows):
https://dl.dropboxusercontent.com/u/1501375/RProject%2BGUI.zip

Anexos

Especificación del fichero 3DTM

El siguiente documento especifica la información que la herramienta RProject guarda en el fichero YAML.

Listado de propiedades del fichero 3DTM

%YAML: Versión de especificación usada para YAML

- *Ejemplo* → %YAML:1.0

date_format: Formato de fecha usada para guardar la fecha

- *Ejemplo* → date_format: mm/dd/yyyy

date: Fecha de creación del fichero 3DTM

- *Ejemplo* → date: "01/16/14"

time_format: Formato de hora usada para guardar la hora

- *Ejemplo* → time_format: "hh:mm:ss"

time: Hora de creación del fichero 3DTM

- *Ejemplo* → time: "16:59:01"

intrinsics: Matriz de intrínsecos de la cámara

rows: Número de filas
cols: Número de columnas
dt: Tipo de datos
data: Valores de la matriz

- *Ejemplo* → intrinsics: !!opencv-matrix
  rows: 3
  cols: 3
  dt: d
  data: [8.1378269547426260e+002, 0., 2.9549548393817958e+002, 0., 8.1272014392646224e+002, 2.6307701792976184e+002, 0., 0., 1.]

image: Nombre de la imagen incluida la extensión

- *Ejemplo* → image: "autodesk_cube_res.png"

model_name: Nombre del modelo incluida la extensión

- *Ejemplo* → model_name: "cube.ply"

model_vertices_format: Formato de los vértices del modelo

- *Ejemplo* → model_vertices_format: "std::vector<cv::Point3f>"

model_vertices_size: Número de vértices en el modelo

- *Ejemplo* → model_vertices_size: 8

model_vertices: Valores de los vértices

- *Ejemplo* → model_vertices: [0., 0., 0., 0., 0., 1., 0., 1., 1., 0., 1., 0., 1., 0.,
  0., 1., 0., 1., 1., 1., 1., 1., 1., 0.]

model_polygons_format: Formato de los polígonos del modelo

- *Ejemplo* → model_polygons_format: "std::vector<std::vector<int>>"

model_polygons_size: Número de polígonos en el modelo

- *Ejemplo* → model_polygons_size: 6

model_polygons_type: Tipo de polígonos del modelo (QUADS/TRIANGLES)

- *Ejemplo* → model_polygons_type: QUADS

model_polygons: Valores de los polígonos

- *Ejemplo* → model_polygons:
  - 0
  - 1
  - 2
  - …..
  - 4
  - 0

keypoints_format: Formato de los keypoints

- *Ejemplo* → keypoints_format: "std::vector<cv::KeyPoint>"

keypoints_size: Número de keypoints

- *Ejemplo* → keypoints_size: 1678

keypoints: Valores de los keypoints

- *Ejemplo* → keypoints: [336., 50., 7., -9.3310943603515625e+001, 40., 0, -1,
  302.,
  57., 7., -9.4139183044433594e+001, 11., 0, -1, 309., 58., 7.,
  …..
  6.5246620178222656e+001, 20., 0, -1, 267., 454., 7.,
  6.9887413024902344e+001, 51., 0, -1]

points3D_format: Formato de los puntos 3D correspondientes a los keypoints

- *Ejemplo* → points3D_format: "std::vector<cv::Point3f>"

points3D_size: Número de puntos 3D correspondientes a los keypoints

- *Ejemplo* → points3D_size: 1678

points3D: Valores de los puntos 3D correspondientes a los keypoints

- *Ejemplo* → points3D: [3.30789424e-002, 9.98188853e-001, 1.74599495e-002, 6.04209974e-002, 9.98247623e-001, 2.06397593e-001, 9.18341503e-002,
  …..
  1.00169051e+000, 9.99673963e-001, 1.29122948e-002, 9.52052951e-001, 9.95823383e-001, 2.98228487e-002, 1.00168025e+000]

descriptors: Matriz con los descriptores de los keypoints

rows: Número de filas
cols: Número de columnas
dt: Tipo de datos
data: Valores de la matriz

- *Ejemplo* → descriptors: !!opencv-matrix
  rows: 1678
  cols: 64
  dt: u
  data: [229, 47, 243, 55, 217, 230, 171, 222, 236, 191, 234, 255, 203, 139, 239, 235, 213, 35, 103, 77, 143, 231, 53, 79, 223, 191,
  …..
  116, 64, 16, 60, 24, 4, 162, 17, 142, 9, 56, 96, 34, 188, 72, 199, 36, 112]

Sistema de carpetas y ficheros

de la

Herramienta RProject

El siguiente documento enumera y describe el sistema de carpetas y ficheros existente de la herramienta RProject para la versión de Windows.

La herramienta es totalmente portable y viene distribuida dentro de un archivo ZIP llamado “RProject+GUI”. Del interior del fichero ZIP se debe descomprimir la carpeta “RProject” donde se considere oportuno. Dicha carpeta es la raíz del sistema de carpetas y ficheros de la herramienta.

En la carpeta “RProject” se localizan dos carpetas:

- ***DATA***: Carpeta donde deben colocarse los ficheros de las imágenes y de los modelos. En esta carpeta la herramienta también guarda durante su ejecución los ficheros finales y temporales, como son las imágenes redimensionadas con las que trabaja, resultados intermedios y los resultados finales.

- ***lib***: Carpeta que contiene el fichero YAML para cargar los parámetros intrínsecos de la cámara.
    - **intrinsics.yml**: Fichero YAML que contiene los parámetros intrínsecos de la cámara. Si se quiere usar otra cámara que tenga parámetros intrínsecos diferentes, debe sobrescribirse el fichero con los nuevos parámetros manteniendo el formato interno del fichero.

ANEXO II

Archivos temporales que se generan en la carpeta DATA durante la ejecución de la herramienta:

- **out.png**: Archivo de imagen que es usado como almacenamiento temporal de imágenes en las distintas fases de procesado de la herramienta.
- **finalMatching.jpg**: Archivo de imagen que es usado como resultado final en el proceso de emparejamiento.
- **points.txt**: Fichero de texto que contiene los puntos 2D y 3D que se han emparejado.
- **3DTM.yml**: Fichero de serialización YAML donde se guarda el modelo 3DTM (*3D Trackable Model*) y otros valores de interés.
- [nombre-imagen]**_res**.[extensión]: Copia del archivo de imagen redimensionado a la resolución de 640X480. "extensión" será la de la imagen utilizada (png, jpg...).

Todos estos archivos son sobrescritos automáticamente durante la ejecución de la herramienta sin guardar copia de ellos.

Archivos finales que se generan en la carpeta DATA durante la ejecución de la herramienta:

- **3DTM(**[nombre-imagen]**).zip**: Fichero comprimido en ZIP que contiene el fichero 3DTM. También incluye la imagen de entrenamiento y el modelo utilizados para generar el modelo 3DTM.

Este archivo se sobrescribe si la imagen de entrenamiento usada en la generación del modelo 3DTM, coincide con la imagen de entrenamiento utilizada en otro modelo 3DTM ya generado.

ANEXO II

Ficheros que se encuentran en la Raíz de la herramienta:

- **Archivos DLL**: Archivos que contienen las librerías de enlace dinámico necesarias para que funcione la herramienta.

- **RProject_ns.exe**: Es el ejecutable principal que permite la ejecución de la herramienta. Se puede usar pasando parámetros o no.

    - Ejecución mediante parámetros:

        RProject_ns.exe [img-entrenamiento] [modelo] [img-test]

- **Frontend.jar**: Interfaz escrita en Java para el manejo de la herramienta.

- **Frontend.exe**: Interfaz Java ejecutada como EXE gracias a JSmooth (Java Executable Wrapper).

A continuación se muestra el árbol de carpetas y ficheros de la herramienta:

RProject

- **DATA**
  - foto1.png
  - foto2.png
  - ….
  - points.txt
  - out.png
  - finalMatching.jpg
  - 3DTM.yml
  - 3DTM(foto1).zip
- **lib**
  - intrinsics.yml

Frontend.exe
Frontend.jar
glut32.dll
opencv_calib3d246.dll
opencv_core246.dll
opencv_features2d246.dll
opencv_flann246.dll
opencv_highgui246.dll
opencv_imgproc246.dll
opengl32.dll
OpenNI.dll
pcl_common_release.dll
pcl_io_ply_release.dll
pcl_io_release.dll
pcl_kdtree_release.dll
pcl_visualization_release.dll
RProject_ns.exe

Leyenda

L: DLLs de la librería OpenCV
L: DLLs de la librería PCL
L: DLLs de la librería GLUT (OpenGL)
L: Fichero de parámetros intrínsecos
L: Ejecutable Java
L: Ejecutable binario en sistema Windows

Planificación

y

Modelo de Desarrollo

Planificación

A continuación se presentan las principales herramientas que se han usado para la realización del proyecto. El objetivo es tener la capacidad de trazar una visión de trabajo que permita realizar un seguimiento del estado del proyecto y detectar las posibles desviaciones durante el mismo.

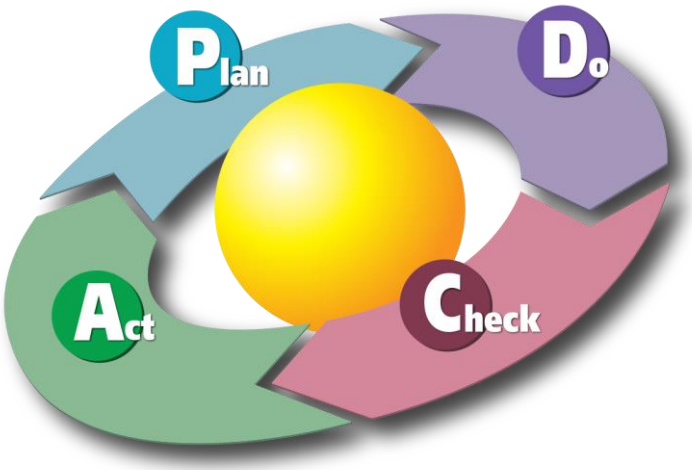


Figura 1: Ciclo PDCA

En la Figura 1 se muestra el ciclo de mejora continua (Plan-Do-Check-Act) aplicado al proyecto. Las acciones de planificar, verificar, y actuar para corregir desviaciones o errores, permiten encauzar los procesos operacionales. Estas acciones englobadas en el ámbito del proceso táctico del proyecto, constituye un ciclo iterativo que se repite durante todo el proyecto.

EDT (Esquema de Descomposición de Trabajo)

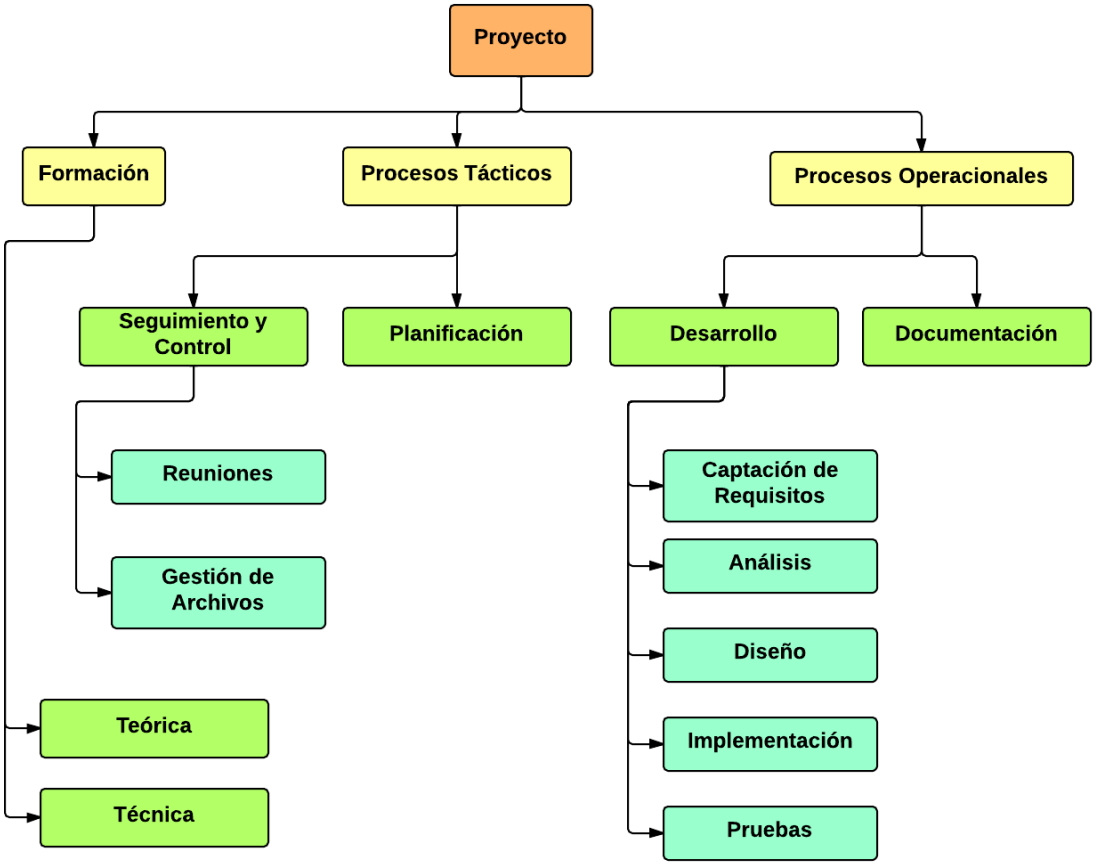


Figura 2: EDT del Proyecto

El EDT presentado la Figura 2 muestra la estructura de trabajo seguida para la consecución del proyecto.

La división del primer nivel (color amarillo) corresponde a una división de los tipos de procesos existentes durante el proyecto que se dividen en tácticos, relacionados con la gestión de proyectos, y operacionales más ligados al plano técnico. Por otro lado se muestra y se separa el proceso formativo, puesto que existe una formación previa en el ámbito teórico y técnico antes de la acometida del proyecto. No obstante el proceso formativo, aunque más intenso en el inicio del proyecto, se mantiene durante todo el proyecto siendo fundamentalmente formación de carácter técnico la que se realiza.

En los subniveles de los procesos tácticos (color verde), se encuentran las dos actividades que posibilitan que se cumpla el ciclo de mejora continua. Se trata del "Seguimiento y Control" y la "Planificación", complementarias entre sí. Lo que la plantificación marca es controlado mediante un seguimiento y un control que validará si existe la necesidad de una rectificación en el proyecto o una replanificación. Las reuniones constituyen el principal procedimiento mediante el cual se valida el desarrollo generado hasta el momento en el proyecto, además de constituir el principal procedimiento de comunicación durante todo el proyecto.

La gestión de archivos hace referencia a todo el código o documentación generada durante el proyecto, así como los procesos de custodia y copia de respaldo de los mismos.

En los subniveles de los procesos operacionales (color verde), se encuentran las actividades de desarrollo y documentación. El proceso documental puede realizarse paralelamente durante el desarrollo y suele ser más recomendable que realizarlo al final del desarrollo. Aunque durante el desarrollo no se haga un proceso documental completo, es importante tomar notas y poner aclaraciones que luego sirvan para generar una mejor documentación.

Las subdivisiones de la actividad de desarrollo (color azul), muestran el ciclo de desarrollo que será aplicado de manera iterativa usando un sistema de prototipado rápido basado en el modelo XP de desarrollo.

La "captación de requisitos" es el procedimiento de entrada que marcará las especificaciones para el desarrollo. El "análisis" es el proceso de refinamiento de las especificaciones y el "diseño" será la materialización del plan de actuación que cumpla las especificaciones. La "implementación" es la codificación del "diseño" utilizando los lenguajes, librería y entorno de desarrollo seleccionados. Las "pruebas" constituyen el proceso de validación de lo implementado frente a las especificaciones marcadas.

Diagrama de Gantt

En la Figura 3 se muestra la planificación y dependencias entre las tareas del proyecto. La planificación está vinculada fuertemente al modelo de desarrollo XP y ofrece una visión complementaria al EDT.

ANEXO III

	Nombre	Duración	Inicio	Fin	Predecesoras
0	**_Planificación RProject**	**85.63d**	**16/09/2013**	**31/01/2014**	
1	**Formación**	**7d**	**16/09/2013**	**24/09/2013**	
2	Preparación del Entorno de Desarrollo	5d	16/09/2013	20/09/2013	
3	CMake	2d	23/09/2013	24/09/2013	2
4	**Tareas**	**58.38d**	**24/09/2013**	**18/12/2013**	**1**
5	**Tarea 1**	**9d**	**24/09/2013**	**07/10/2013**	
6	Captación de Requisitos	3h	24/09/2013	25/09/2013	
7	Análisis	6h	25/09/2013	25/09/2013	6
8	Diseño	2h	25/09/2013	26/09/2013	7
9	Implementación	7d	26/09/2013	07/10/2013	8
10	Pruebas	5h	07/10/2013	07/10/2013	9
11	**Tarea 2**	**12d**	**07/10/2013**	**23/10/2013**	**5**
12	Captación de Requisitos	3h	07/10/2013	08/10/2013	
13	Análisis	10h	08/10/2013	09/10/2013	12
14	Diseño	2h	09/10/2013	09/10/2013	13
15	Implementación	9d	09/10/2013	22/10/2013	14
16	Pruebas	9h	22/10/2013	23/10/2013	15
17	**Tarea 3**	**3.88d**	**23/10/2013**	**30/10/2013**	**11**
18	Captación de Requisitos	3h	23/10/2013	24/10/2013	
19	Análisis	7h	24/10/2013	24/10/2013	18
20	Diseño	2h	28/10/2013	28/10/2013	19
21	Implementación	2d	28/10/2013	30/10/2013	20
22	Pruebas	3h	30/10/2013	30/10/2013	21
23	**Tarea 4**	**2.63d**	**30/10/2013**	**04/11/2013**	**17**
24	Captación de Requisitos	3h	30/10/2013	30/10/2013	
25	Análisis	4h	30/10/2013	31/10/2013	24
26	Diseño	2h	31/10/2013	31/10/2013	25
27	Implementación	1d	31/10/2013	04/11/2013	26
28	Pruebas	4h	04/11/2013	04/11/2013	27
29	**Tarea 5**	**6.25d**	**04/11/2013**	**13/11/2013**	**23**
30	Captación de Requisitos	3h	04/11/2013	05/11/2013	
31	Análisis	8h	05/11/2013	06/11/2013	30
32	Diseño	2h	06/11/2013	06/11/2013	31
33	Implementación	4d	06/11/2013	12/11/2013	32
34	Pruebas	5h	12/11/2013	13/11/2013	33
35	**Tarea 6**	**9.5d**	**13/11/2013**	**26/11/2013**	**29**
36	Captación de Requisitos	5h	13/11/2013	13/11/2013	
37	Análisis	10h	13/11/2013	14/11/2013	36
38	Diseño	3h	14/11/2013	15/11/2013	37
39	Implementación	6d	15/11/2013	25/11/2013	38
40	Pruebas	10h	25/11/2013	26/11/2013	39
41	**Tarea 7**	**6d**	**26/11/2013**	**04/12/2013**	**35**
42	Captación de Requisitos	3h	26/11/2013	26/11/2013	
43	Análisis	5h	27/11/2013	27/11/2013	42
44	Diseño	2h	27/11/2013	27/11/2013	43
45	Implementación	4d	27/11/2013	03/12/2013	44
46	Pruebas	6h	03/12/2013	04/12/2013	45
47	**Tarea 8**	**2.25d**	**04/12/2013**	**09/12/2013**	**41**
48	Captación de Requisitos	3h	04/12/2013	04/12/2013	
49	Análisis	6h	04/12/2013	05/12/2013	48
50	Diseño	2h	05/12/2013	05/12/2013	49
51	Implementación	2h	05/12/2013	09/12/2013	50
52	Pruebas	5h	09/12/2013	09/12/2013	51
53	**Tarea 9**	**6.88d**	**09/12/2013**	**18/12/2013**	**47**
54	Captación de Requisitos	3h	09/12/2013	10/12/2013	
55	Análisis	8h	10/12/2013	10/12/2013	54
56	Diseño	2h	11/12/2013	11/12/2013	55
57	Implementación	4d	11/12/2013	17/12/2013	56
58	Pruebas	10h	17/12/2013	18/12/2013	57
59	**Interfaz**	**2d**	**18/12/2013**	**20/12/2013**	**4**
60	Captación de Requisitos	1h	18/12/2013	18/12/2013	
61	Análisis	2h	18/12/2013	18/12/2013	
62	Diseño	5h	18/12/2013	19/12/2013	
63	Implementación	2d	18/12/2013	20/12/2013	
64	Pruebas	6h	18/12/2013	19/12/2013	
65	Documentación	17d	07/01/2014	31/01/2014	

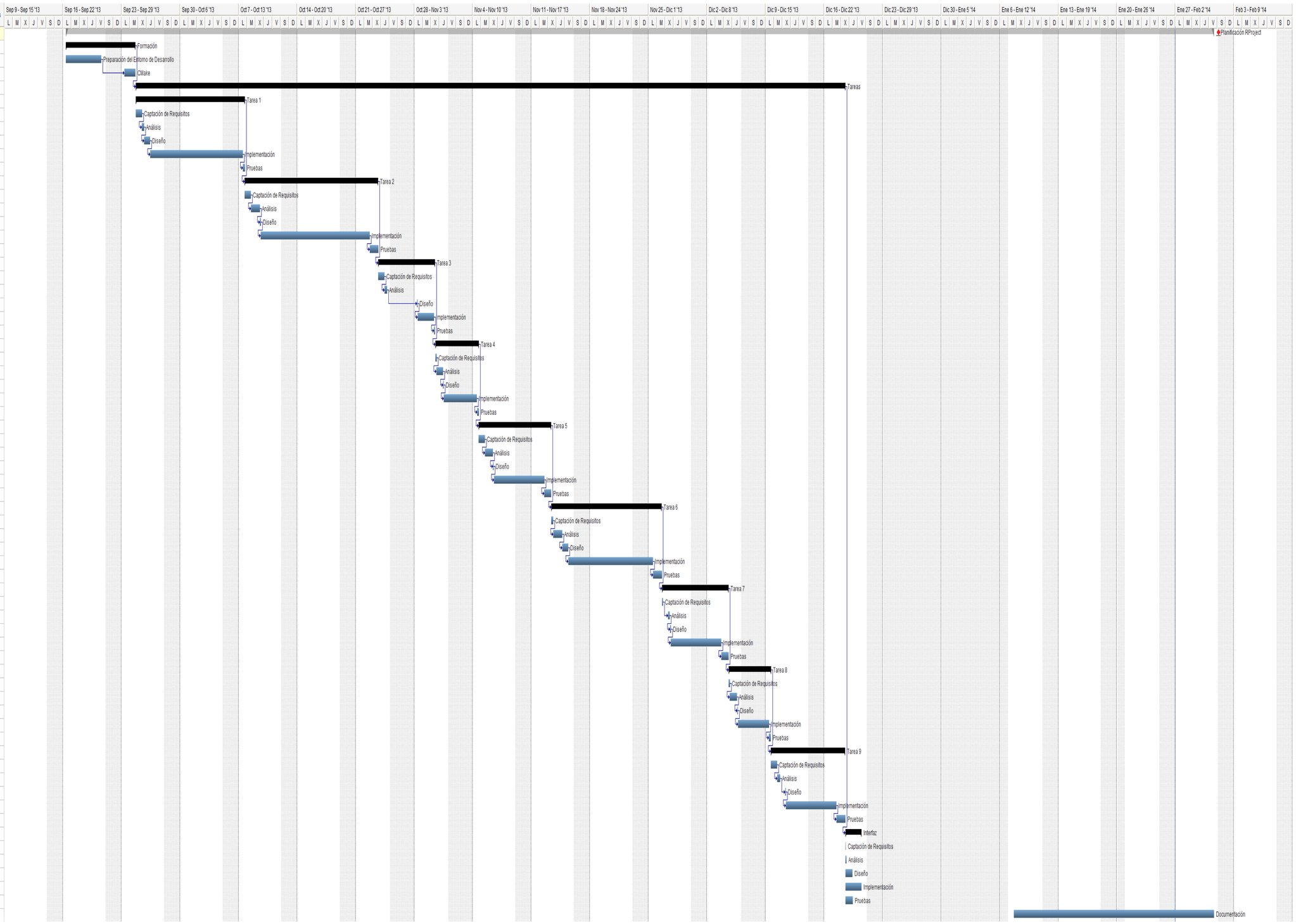


Figura 3: Planificación

Modelo de Desarrollo

La metodología empleada para el desarrollo se engloba en las denominadas "metodologías ágiles", debido a que durante el desarrollo del proyecto se genera software funcional en un plazo de tiempo relativamente corto, se revisa continuamente para aplicar los cambios o corregir fallos, y se adapta el producto a las exigencias finales. El tipo de desarrollo es motivado por la búsqueda de maximizar resultados, aportando la mayor eficiencia posible durante el proceso.

El modelo esencial de desarrollo que se sigue, es un modelo de Programación Extrema (XP), el cual se centra en ir desarrollando pequeños paquetes incrementales denominados módulos o incrementos (véase Figura 4).

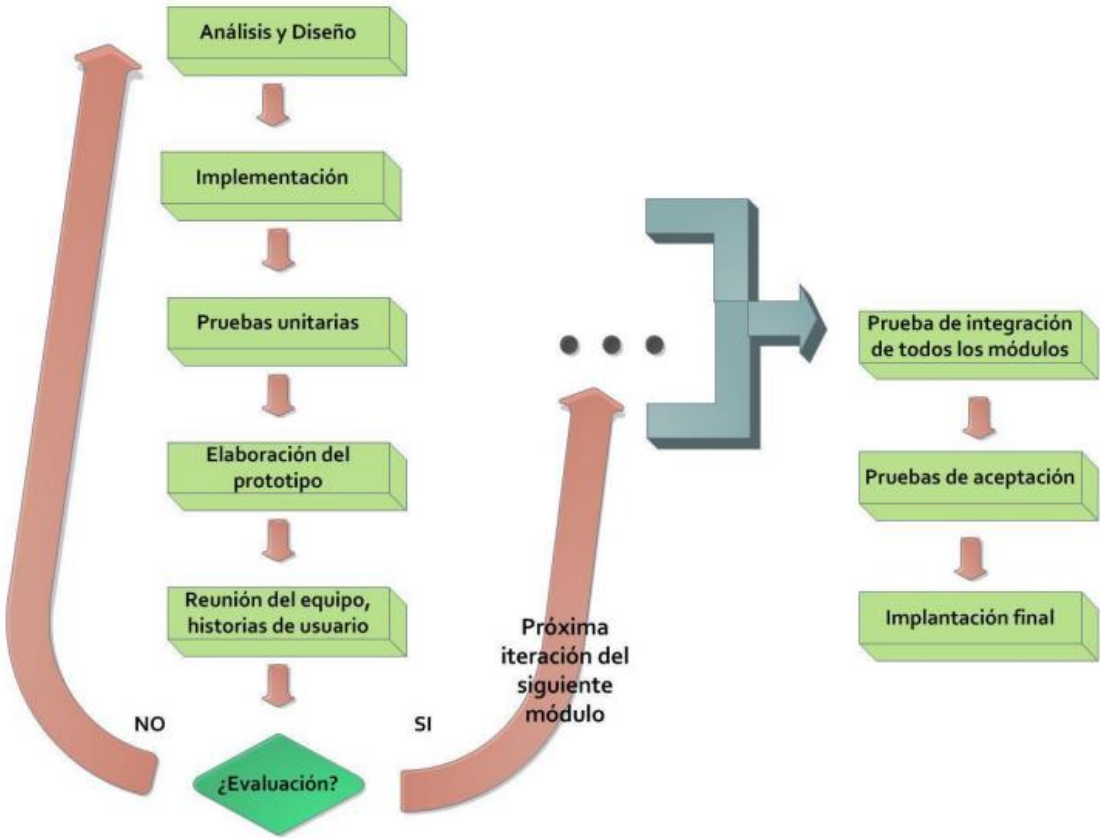


Figura 4: Esquema de Desarrollo con XP

Estos módulos representan pequeños incrementos de la funcionalidad de la aplicación final, escalándose continuamente la aplicación mediante iteraciones en las que se evalúa y acepta (o no se acepta) el incremento. Esta forma de trabajo está ligada con un sistema de prototipado no destructivo, los cuales son funcionales, pero no del todo eficientes, y que pasarán a formar parte de la aplicación final en caso de ser evaluados y aceptados tras la conveniente refactorización del código.

ANEXO III

El modelo de desarrollo de Programación Extrema XP se caracteriza por:

- Simplicidad en el código
- Desarrollo iterativo e incremental
- Retroalimentación (Feedback)
- Pruebas unitarias
- Corrección de errores y refactorización del código

La simplicidad es la base de la programación XP. Se simplifica el diseño para agilizar el desarrollo y facilitar el mantenimiento, puesto que un diseño complejo del código junto a sucesivas modificaciones, por parte de diferentes desarrolladores, hacen que la complejidad aumente exponencialmente.

Para mantener la simplicidad es necesaria la refactorización del código siempre que sea posible y casi obligatorio cuando un incremento es validado y aprobado. Ésta es la manera de mantener el código simple a medida que crece.

También se aplica la simplicidad en la documentación, de esta manera el código debe comentarse en su justa medida, intentando que el código esté autodocumentado. Para ello se deben elegir adecuadamente los nombres de las variables, métodos y clases. Los nombres largos no decrementan la eficiencia del código ni el tiempo de desarrollo gracias a las herramientas de autocompletado y refactorización que existen actualmente.

El uso de sistemas de generación de documentación mediante anotaciones a pie código, como puede ser Doxygen o JavaDoc, permiten además de una mejor autodocumentación del código, la generación de documentación reglada y ordenada de manera automática. La definición de una guía de documentación para el código, mejora además la homogeneidad de la misma. Véase el anexo V.

Una de las ventajas de tener el cliente integrado en el proyecto, es conocer en tiempo real su opinión sobre el estado del proyecto. Esto constituye un valioso feedback y aumenta la satisfacción general del cliente al conocer el estado continuado del proyecto.

Al realizarse ciclos muy cortos tras los cuales se muestran resultados, se minimiza el tener que rehacer partes que no cumplen con los requisitos y ayuda al programador a centrarse en lo que es más importante. De esta forma se evitan semanas de trabajo que pueden tirarse por la borda debido a cambios en los criterios del cliente o malentendidos por parte del equipo de desarrollo.

Todo lo expuesto, en la práctica implica siempre diseñar y programar para hoy y no para mañana, evitando empantanarse en el diseño y requerir demasiado tiempo y trabajo para implementar el resto del proyecto. El efecto evidente es que el desarrollador se sienta cómodo para reconstruir el código cuando sea necesario, revisando el código existen o

desechándolo si fuese necesario. Por otro lado, la ejecución frecuentemente de pruebas, informa sobre el estado de salud del código y permite descubrir fallos.

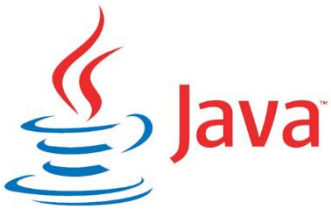
Este tipo de desarrollo aplicando esta metodología aporta flexibilidad y tolerancia a cambios, muy necesarios cuando se está investigando y es necesario deshacerse de posibles líneas de desarrollo y adoptar otras más factibles y viables.

Lenguajes, librerías y herramientas utilizadas para el desarrollo

C++ es un lenguaje de programación diseñado a mediados de los años 1980 por Bjarne Stroustrup. La intención de su creación fue el extender al exitoso lenguaje de programación C con mecanismos que permitan la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, el C++ es un lenguaje híbrido.

Posteriormente se añadieron facilidades de programación genérica, que se sumó a los otros dos paradigmas que ya estaban admitidos (programación estructurada y la programación orientada a objetos). Por esto se suele decir que el C++ es un lenguaje de programación multiparadigma.

Actualmente existe un estándar, denominado ISO C++, al que se han adherido la mayoría de los fabricantes de compiladores más modernos.



El lenguaje de programación Java fue originalmente desarrollado por James Gosling de Sun Microsystems (la cual fue adquirida por la compañía Oracle) y publicado en 1995 como un componente fundamental de la plataforma Java de Sun Microsystems. Su sintaxis deriva mucho de C y C++, pero tiene menos facilidades de bajo nivel que cualquiera de ellos. Las aplicaciones de Java son generalmente compiladas a bytecode (clase Java) que puede ejecutarse en cualquier máquina virtual Java (JVM) sin importar la arquitectura de la computadora subyacente.

Es un lenguaje de programación de propósito general, concurrente, orientado a objetos y basado en clases que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo.

OpenCV es una biblioteca libre de visión artificial originalmente desarrollada por Intel. Desde que apareció su primera versión alfa en el mes de enero de 1999, se ha utilizado en infinidad de aplicaciones. Desde sistemas de seguridad con detección de movimiento, hasta aplicativos de control de procesos donde se requiere reconocimiento de objetos. Esto se debe a que su publicación se da bajo licencia BSD, que permite que sea usada

libremente para propósitos comerciales y de investigación con las condiciones en ella expresadas.

OpenCV es multiplataforma, existiendo versiones para GNU/Linux, Mac OS X y Windows. Contiene más de 500 funciones que abarcan una gran gama de áreas en el proceso de visión, como reconocimiento de objetos (reconocimiento facial), calibración de cámaras, visión estéreo y visión robótica.

El proyecto pretende proporcionar un entorno de desarrollo fácil de utilizar y altamente eficiente.

PCL es una librería independiente de código abierto que incluye numerosos algoritmos avanzados para nubes de puntos 3D y procesamiento de la geometría. La librería contiene algoritmos para el filtrado, la estimación de características, la reconstrucción de superficies, el registro, el modelo adecuado y la segmentación. PCL es desarrollado por un gran consorcio de investigadores e ingenieros de todo el mundo. Está escrito en C++ y liberado bajo la licencia BSD. Estos algoritmos se pueden utilizar, por ejemplo, en la percepción de la robótica para reconocer objetos en el mundo en base a su apariencia geométrica, y crear superficies a partir de nubes de puntos y visualizarlos.

GLUT (del inglés OpenGL Utility Toolkit) es una biblioteca de utilidades para programas OpenGL que proporciona diversas funciones de entrada/salida con el sistema operativo. Entre las funciones que ofrece se incluyen declaración y manejo de ventanas y la interacción por medio de teclado y ratón. También posee rutinas para el dibujado de diversas primitivas geométricas (tanto sólidas como en modo wireframe) que incluyen cubos, esferas y teteras. También tiene soporte para creación de menús emergentes.

La versión original de GLUT fue escrita por Mark J. Kilgard, autor de "*OpenGL Programming for the X Window System y The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics"*, mientras trabajaba para Silicon Graphics.

Los dos objetivos de GLUT son permitir la creación de código más portable entre diferentes sistemas operativos (GLUT es multiplataforma) y hacer OpenGL más simple. Introducirse en la programación con OpenGL utilizando GLUT conlleva normalmente sólo unas pocas líneas de código y hace innecesario el conocimiento de las APIs específicas de cada sistema operativo.

Info-ZIP es un software de código abierto para manejar archivos ZIP que se creó en 1989. Existen versiones de la aplicación para Windows, Mac y Linux. Al tener acceso al código se puede incrustar Info-Zip en cualquier desarrollo como si de una librería se tratase.

Microsoft Visual Studio es un entorno de desarrollo integrado (IDE, por sus siglas en inglés) para sistemas operativos Windows. Soporta múltiples lenguajes de programación tales como C++, C#, Visual Basic .NET, F#, Java, Phyton, Ruby, php; al igual que entornos de desarrollo web como ASP.NET MVC, Django, et., a lo cual sumarle las nuevas capacidades online bajo Windows Azure en forma del editor Monaco.

Visual Studio permite a los desarrolladores crear aplicaciones, sitios y aplicaciones web, así como servicios web en cualquier entorno que soporte la plataforma .NET (a partir de la versión .NET 2002). Así se pueden crear aplicaciones que se comuniquen entre estaciones de trabajo, páginas web, dispositivos móviles, dispositivos embebidos, consolas (la Xbox 360 y Xbox One), etc.

Eclipse es un programa informático compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse). Sin embargo, también se puede usar para otros tipos de aplicaciones cliente, como BitTorrent o Azureus.

Eclipse es también una comunidad de usuarios, extendiendo constantemente las áreas de aplicación cubiertas. Un ejemplo es el recientemente creado Eclipse Modeling Project, cubriendo casi todas las áreas de Model Driven Engineering.

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

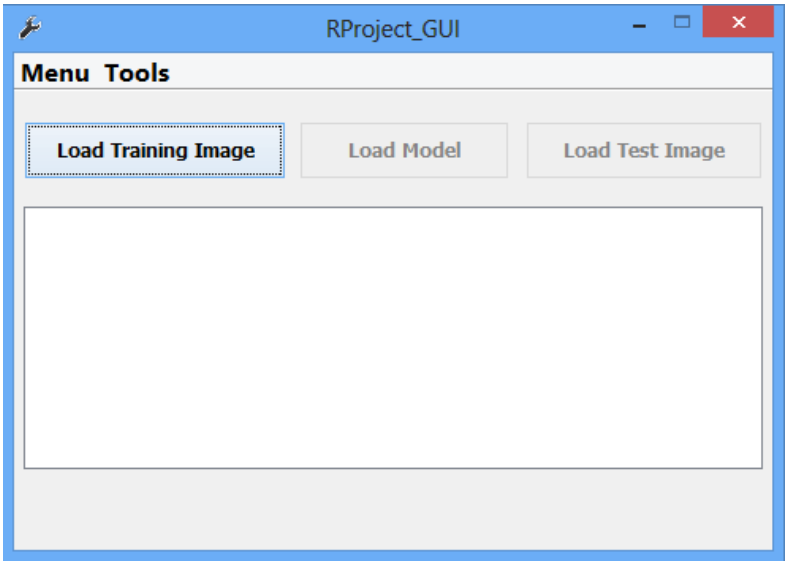
Manual de Usuario de la Herramienta RProject

El siguiente documento describe el funcionamiento de la herramienta RProject usando interfaz gráfica y en un sistema operativo Windows. La herramienta RProject permite generar *3D Trackable Models* (3DTM) y verificar su funcionamiento. Los términos y procesos aquí indicados se explican en la memoria.

La herramienta puede ser usada con una interfaz escrita en Java disponible en versión JAR y en versión EXE (*Frontend.jar* y *Frontend.exe*). Ambas son ejecutables mediante una selección de dos clicks de ratón. Para que la interfaz se comunique con la herramienta, debe ejecutarse la interfaz estando en la misma carpeta que el ejecutable de la herramienta (*RPorject_ns.exe*).

La versión EXE resulta una versión mucho más integrada con el entorno de Windows y se presenta la aplicación con el siguiente icono.

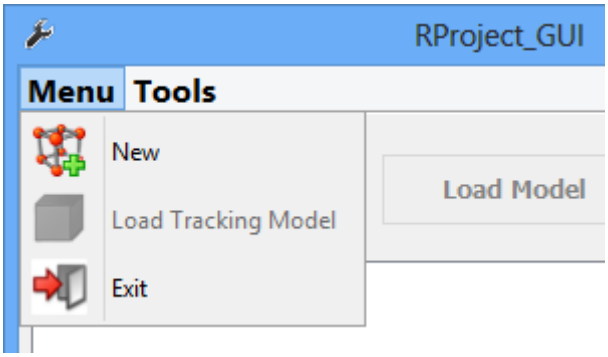
Una vez se selecciona cualquiera de las dos versiones de la interfaz se mostrará la siguiente ventana.



En la ventana se puede observar en la parte superior la barra de menú. Inmediatamente debajo se encuentran los botones de selección de imágenes y modelo. En el centro de la ventana se puede apreciar un contenedor en blanco que mostrará información textual del estado de la herramienta. En ocasiones, por falta de actividad en la herramienta, aparecerá el siguiente icono animado.

Este icono muestra que la herramienta sigue trabajando, pero todavía no tiene la información requerida. Además sirve para saber si la aplicación ha quedado colgada al no moverse el icono o al no desaparecer.

El “Menu” de la interfaz dispone de las siguientes opciones seleccionables por medio del ratón.

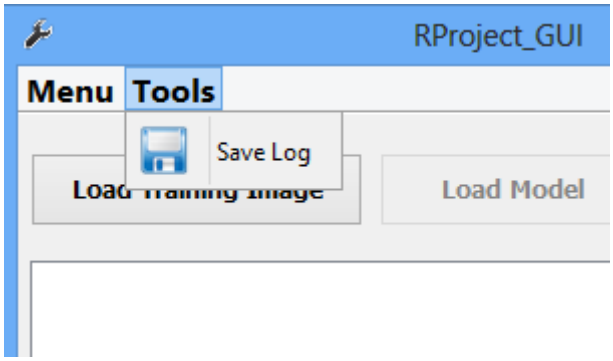


 **New**: Vuelve a iniciar el proceso de creación de un 3DTM

 **Load Tracking Model**: Permite cargar un 3DTM para verificar su funcionamiento con una imagen de Test (No disponible actualmente).

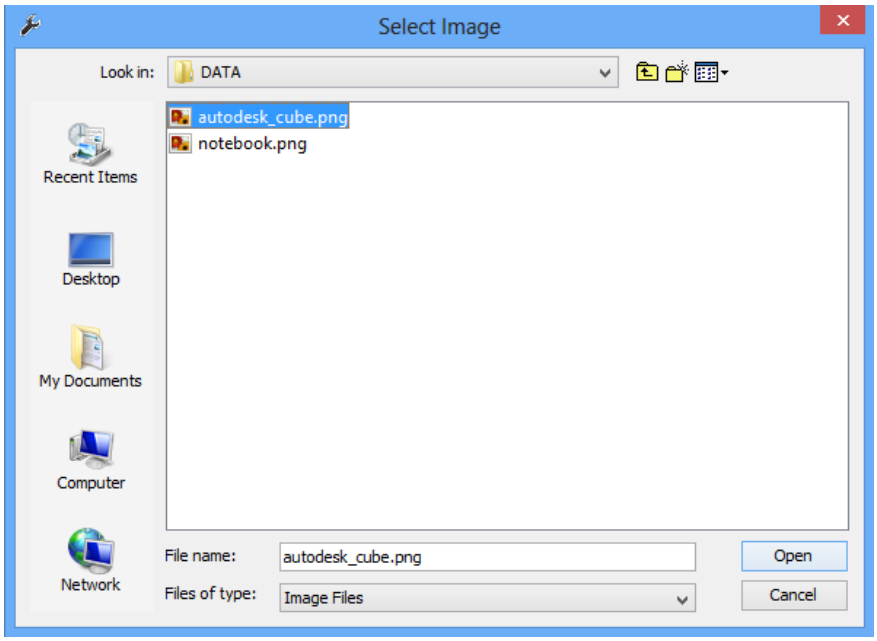
 **Exit**: Salir de la herramienta. Causa el mismo efecto que cerrar con el aspa roja de la parte superior derecha de la ventana.

En la sección “Tools” de la interfaz se dispone de la siguiente opción:



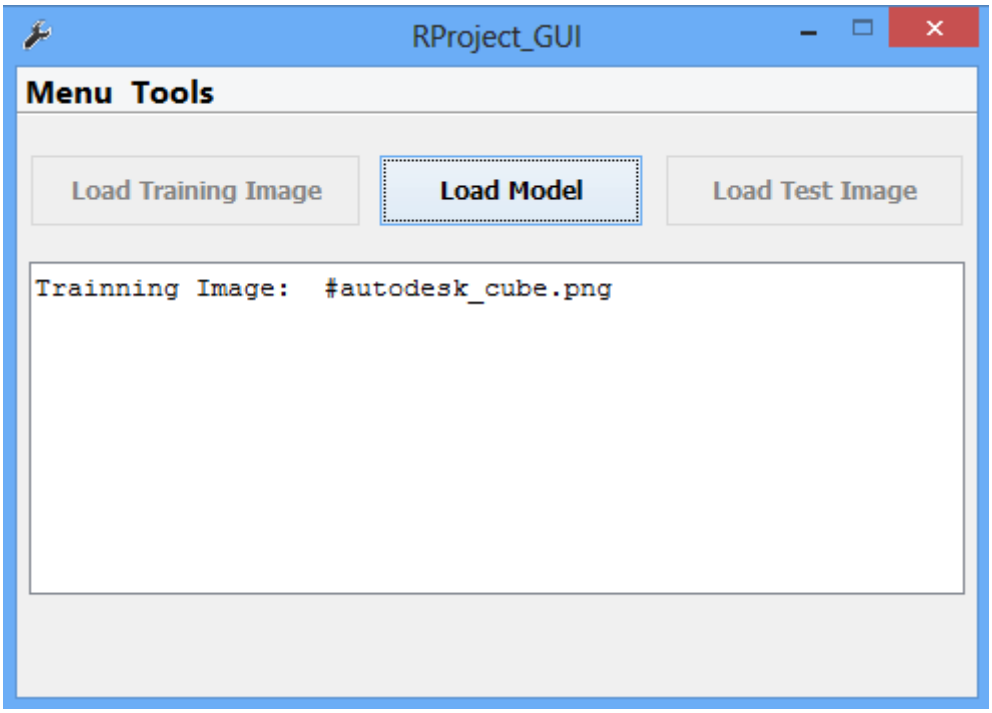
 **Save Log**: Permite guardar todo el Log creado durante la generación de un 3DTM. Los mensajes de Log son mostrados en el contenedor blanco del centro de la ventana.

Para crear un 3DTM lo primero que hay que hacer es elegir la imagen de entrenamiento seleccionado el botón “Load Tranning Image”. Una vez se selecciona la opción, se abre una ventana para escoger un archivo de tipo imagen de la carpeta “DATA” de la herramienta.

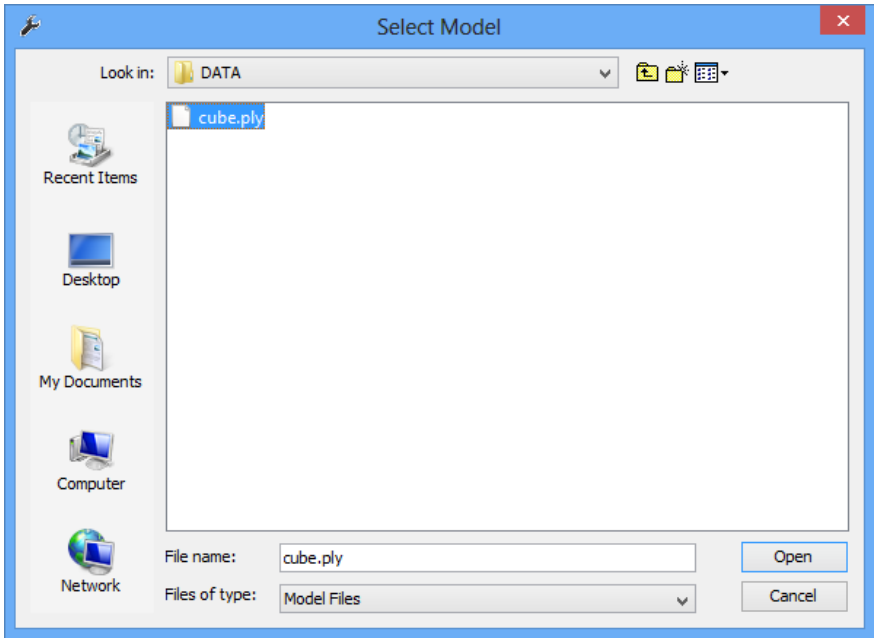


Se selecciona el archivo de imagen deseado haciendo doble click sobre el fichero o seleccionando el archivo y después seleccionando abrir.

Tras seleccionar la imagen de entrenamiento, se desactivará la opción “Load Tranning Image” y se activará la opción “Load Model” para poder ser seleccionada.

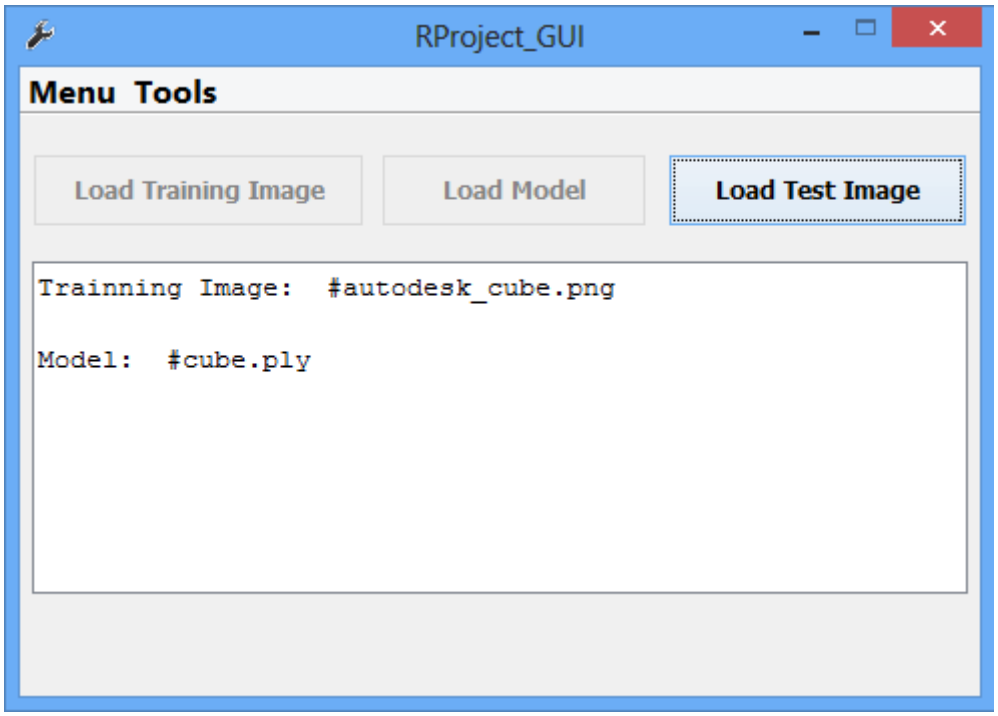


Al seleccionar “Load Model” se abre una ventana para escoger el modelo de la carpeta “DATA” de la herramienta.

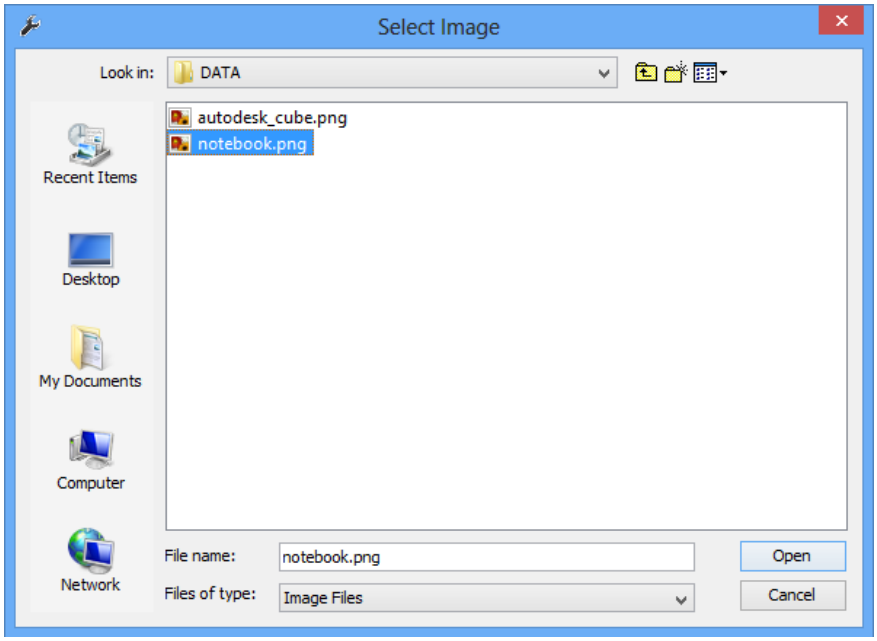


Se selecciona el modelo deseado haciendo doble click sobre el fichero o seleccionando el archivo y después seleccionando abrir.

Tras seleccionar el modelo, se desactivará la opción “Load Model” y se activará la opción “Load Test Image” para poder ser seleccionada.

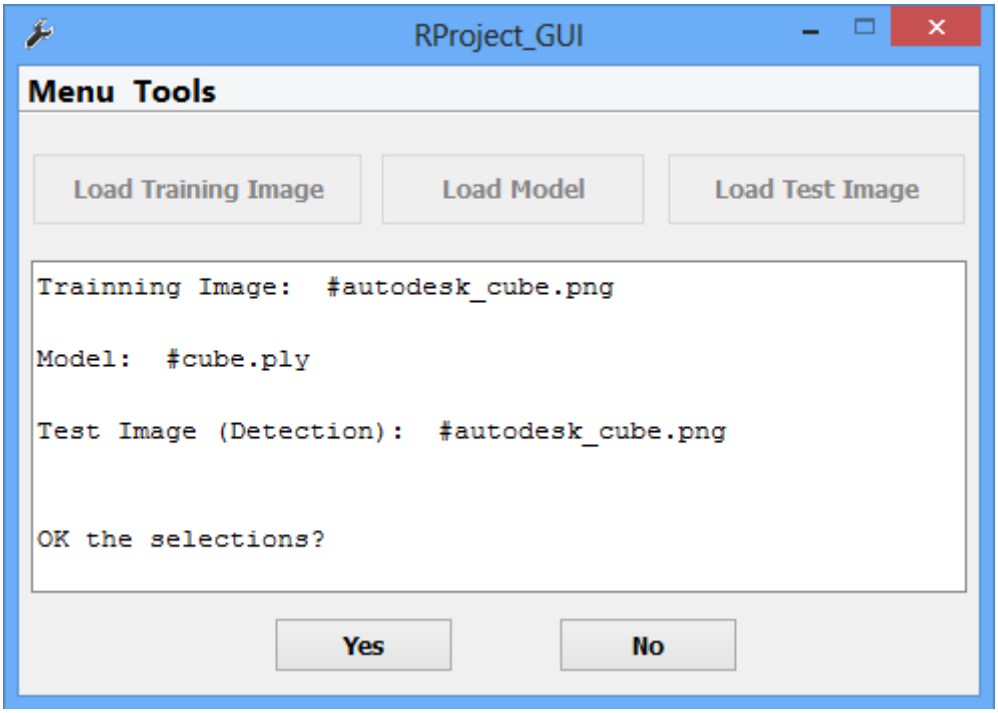


Tras seleccionar “Load Test Image” se abre una ventana para escoger un archivo de tipo imagen de la carpeta “DATA” de la herramienta.



Se selecciona el archivo de imagen deseado haciendo doble click sobre el fichero o seleccionando el archivo y después seleccionando abrir.

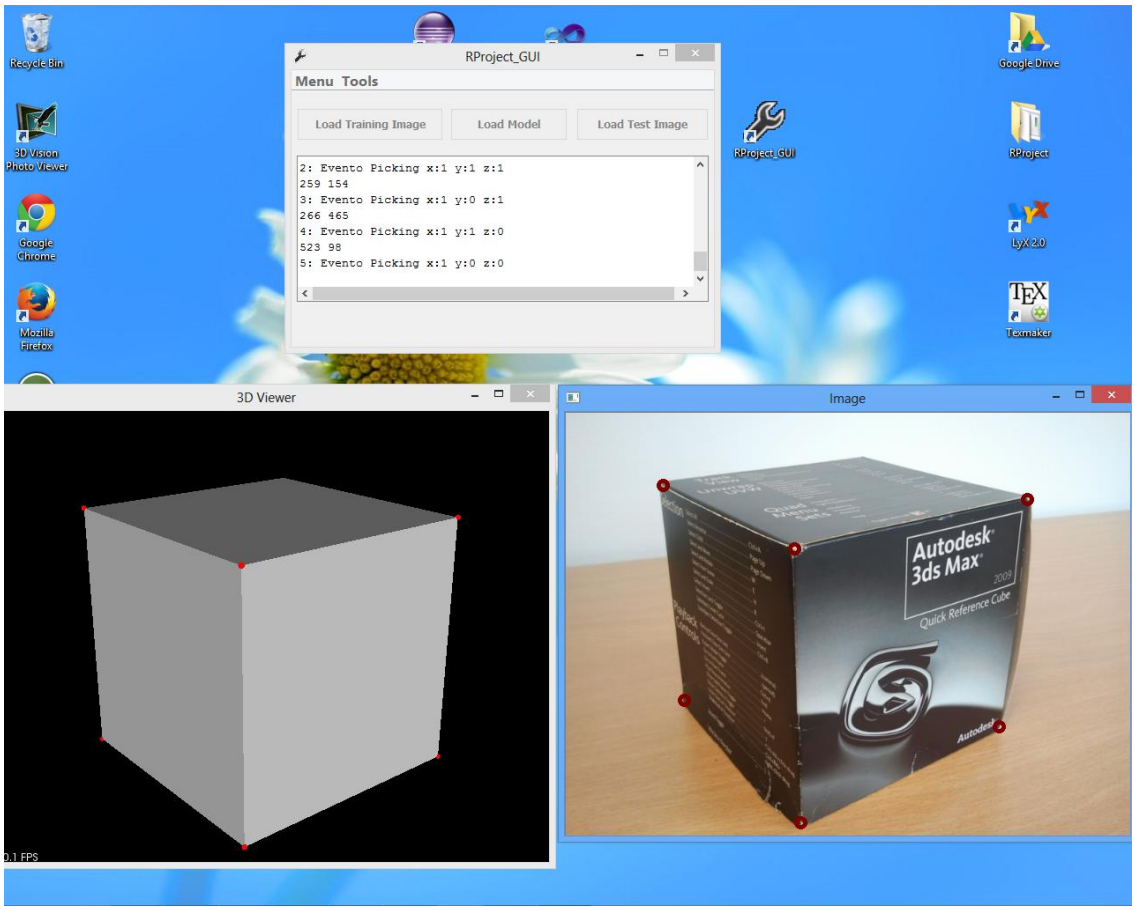
Una vez hechas las selecciones, la herramienta preguntará si las selecciones son correctas. En caso de serlo seleccionar “Yes” en caso contrario “No”.



Si se selecciona “Yes” se pasará al siguiente paso de la herramienta. En caso de seleccionar “No” se volverá a realizar la selección de la imagen de entrenamiento, el modelo y la imagen de prueba.

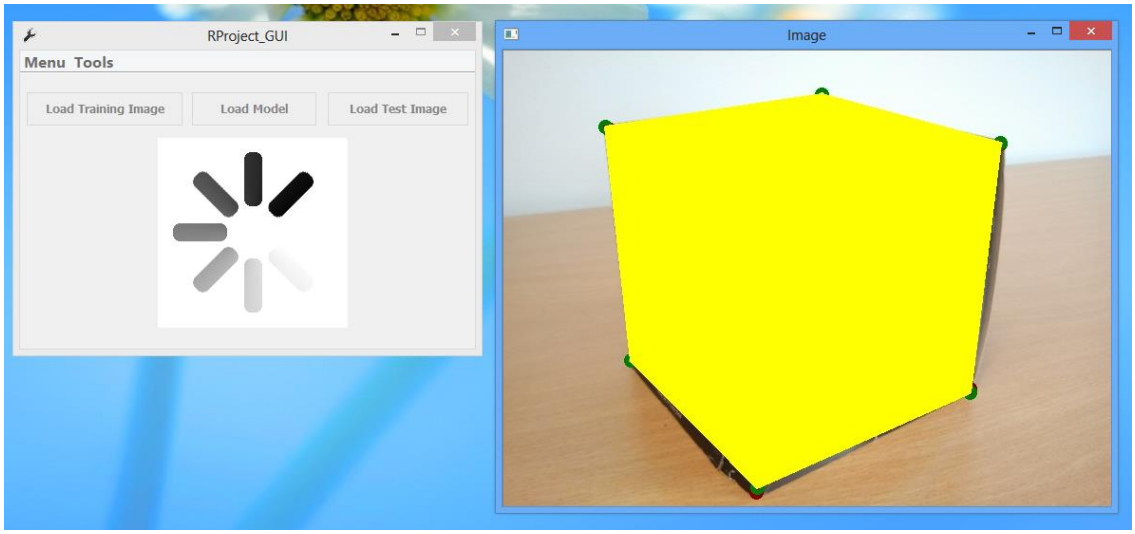
Si confirmamos que lo seleccionado es correcto, se nos abrirán dos ventanas nuevas. Una con la imagen de entrenamiento y otra con el modelo. Pulsando la tecla “shift” y seleccionando con el ratón podremos ir emparejando los vértices del modelo con los vértices del objeto de la imagen.

Se puede girar el modelo, para la selección de vértices. La forma de hacerlo es seleccionando un punto cualquiera del modelo y arrastrando el ratón hacia el borde de la ventana mientras se mantiene pulsada la tecla de selección del ratón.

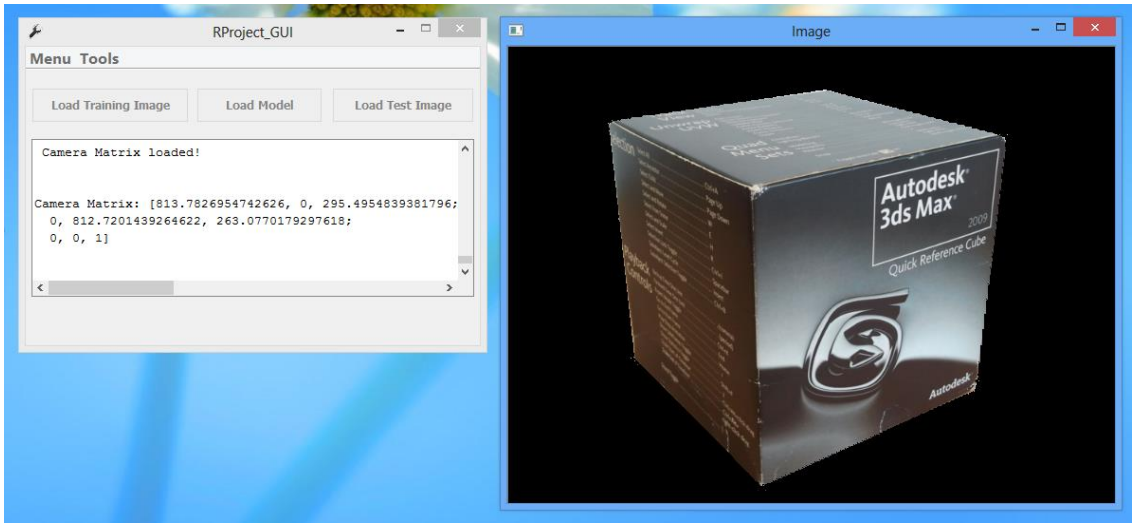


Se precisa un mínimo de 4 emparejamiento de puntos para que la herramienta prosiga su ejecución. Cuando se termine de realizar la selección de emparejamientos, el usuario podrá cerrar ambas ventana. Entonces la ejecución pasará al siguiente paso.

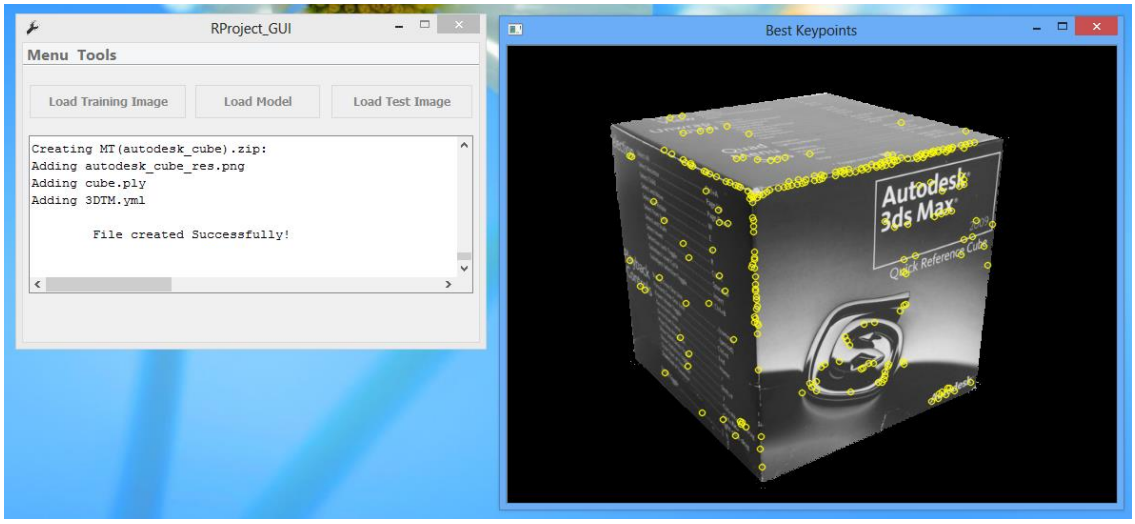
Tras cerrar las ventanas nos aparecerá el modelo proyectado en la imagen en una nueva ventana. Si el proceso ha funcionado correctamente el modelo aparecerá superpuesto al objeto de la imagen.



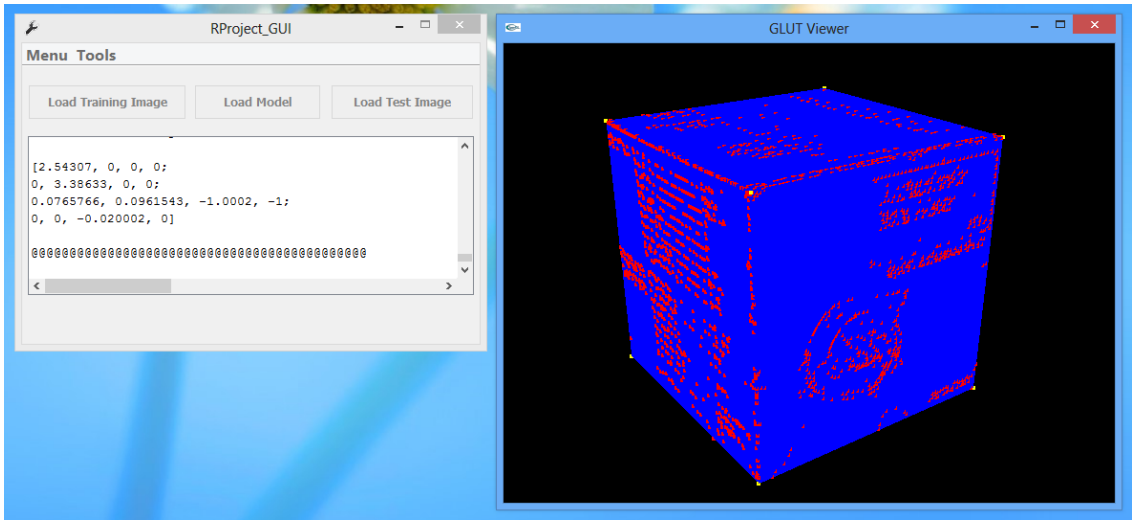
Para continuar la ejecución cerramos la ventana y nos aparecerá otra con el objeto de la imagen aislado del resto de la fotografía.



La ejecución continúa cerrando la ventana, tras lo cual se abre otra ventana con los puntos de interés detectado en la superficie del modelo.

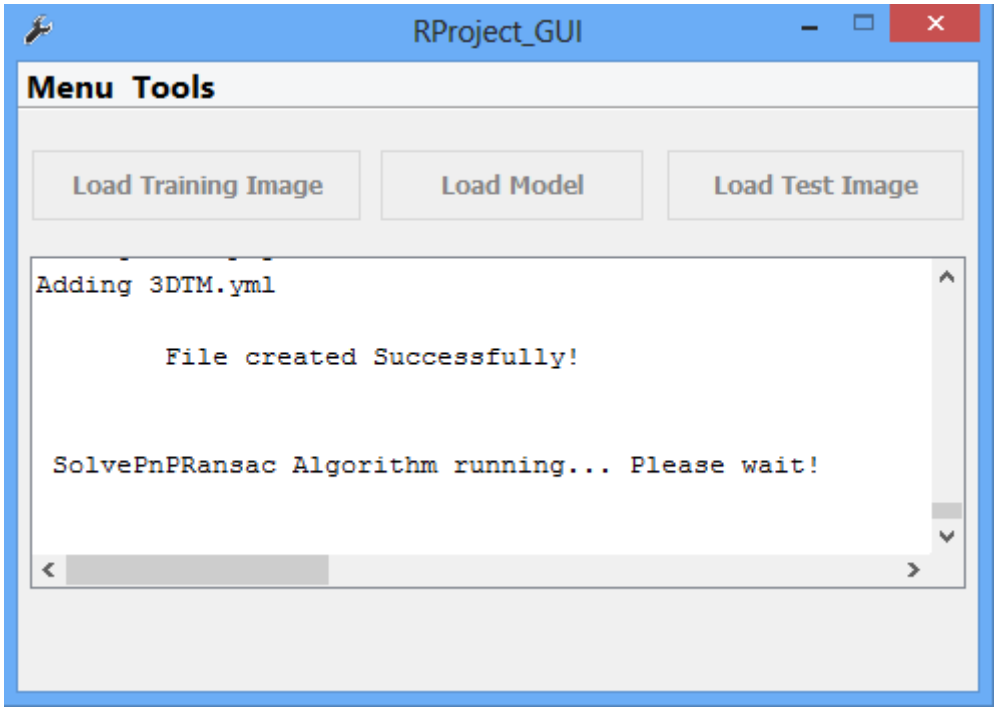


Cerrando la ventana, se iniciará el cálculo de la coordenadas 3D para los puntos de interés detectados. Cuando el cálculo termina se abre una ventana con el modelo 3D y los puntos de interés en su superficie.

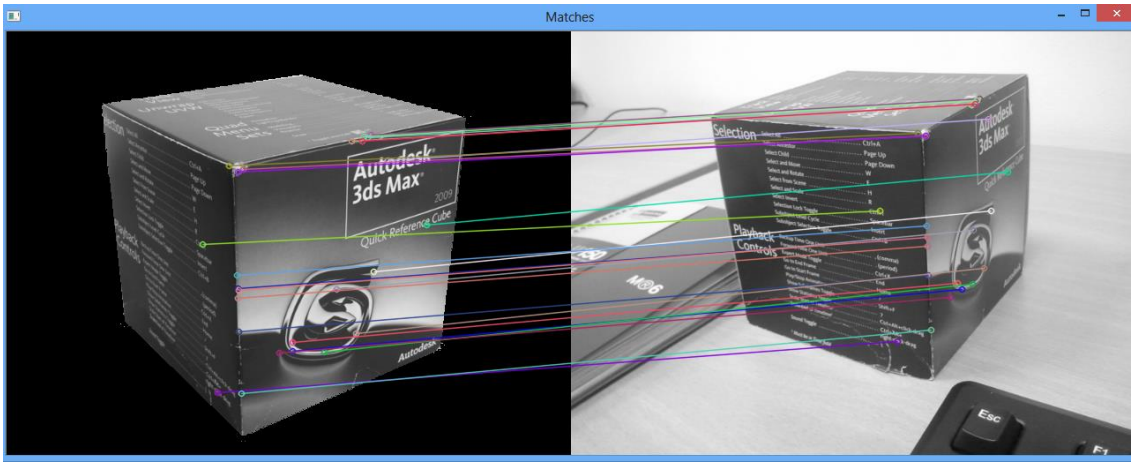


Esta vez no cerraremos la ventana desde el aspa. Usaremos la tecla “ESC” del teclado para salir. Si se intenta cerrar la ventana desde el aspa roja de la ventana, la ejecución continúa igual que pulsando la tecla “ESC”, pero no desaparece la ventana hasta finalizar la ejecución.

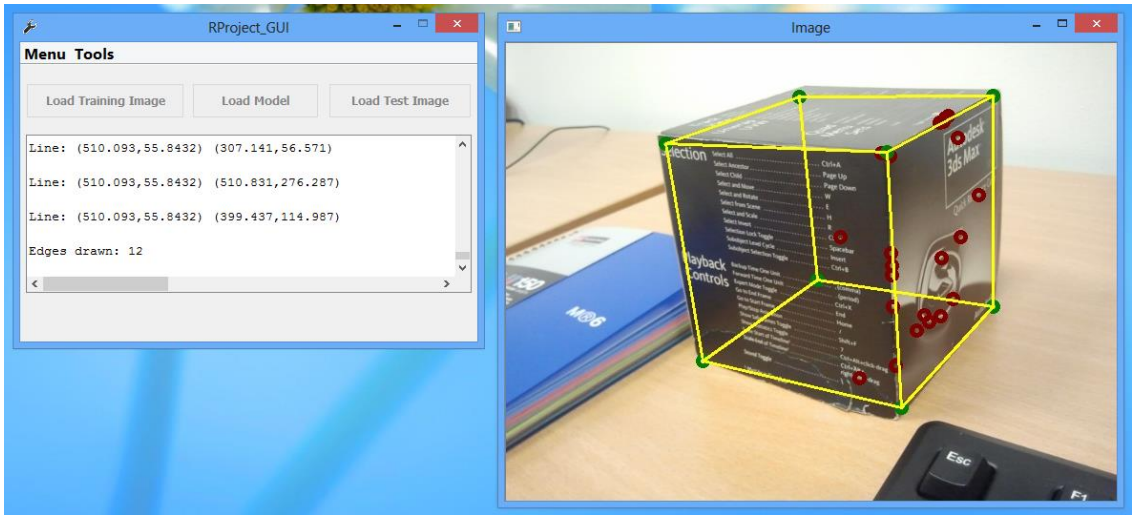
Tras pulsar “ESC” en el teclado, la herramienta guarda el modelo 3DTM en un fichero ZIP e inicia el proceso de cálculo para la detección del modelo 3DTM en la imagen de prueba. El fichero ZIP es guardado en la carpeta “DATA” de la herramienta.



Tras un tiempo de espera para el cálculo, se muestra una ventana con los emparejamientos entre la imagen de prueba y el modelo 3DTM, representado por la imagen de entrenamiento.



Al cerrar la ventana, se abrirá otra que nos posiciona el 3DTM sobre el objeto de la imagen de prueba.



Se presenta de color amarillo las aristas del modelo, de color verde los vértices del modelo y de color rojo los puntos de interés detectado en la imagen de prueba.

Al cerrar la ventana la ejecución de la aplicación habrá concluido. Es aconsejable guardar el Log en este momento, si se desea guardar toda la información del proceso (Tools → Save Log). El fichero de Log queda almacenado en un fichero de texto en la carpeta "RProject", donde se encuentra la interfaz y el ejecutable de la herramienta.

Para iniciar el proceso de nuevo seleccionar "New" (Menu → New). Si se desea salir, cerrar las ventanas o seleccionar "Exit" (Menu → Exit).

Doxygen: Propuesta de estilo para Vicomtech

- No usar tildes ni ñ para evitar problemas de pérdida de símbolos en la documentación final.
- El formato usado para la documentación en la medida de lo posible coincidirá con el usado en JavaDoc y que además es soportado por Doxygen:

```
/**
*       @tag1
*       @tag2
*/
```

- Existen una serie de "markdowns" que permiten salvar ciertos obstáculos con el estilo del texto:

    - ****texto en negrita****
    - Cualquier texto subrayado con ---------- es un título de segundo nivel. Doble subrayado es un título de primer nivel (Mayor tamaño).

```
Title level 1
---------------
---------------

Title level 2
---------------
```

    - Línea horizontal - - - (Con espacios entre los guiones)
    - Tablas

```
First Header  | Second Header
------------- | -------------
Content Cell  | Content Cell
Content Cell  | Content Cell
```

    - Saltos de línea con \n o dejando una línea vacía.
    - Texto en crudo

```
~~~~~~~~~~~~~~~~
5      3      7
4      1      4
8      5      9
~~~~~~~~~~~~~~~~
```

Tags más relevantes

- @brief → Da un descripción
- @author → Establece la autoría del código
- @since → Permite saber cuál es la última vez que se modificó el código
- @version → Permite saber qué versión tiene el código
- @see → Establece una referencia cruzada
- @note → Documenta una nota
- @param[in/out] → Documenta los parámetros pasado a una función
- @return → Documenta el valor de retorno de una función
- @pre → Documenta las precondiciones de una función
- @post →Documenta las precondiciones de una función
- @struct →Indica que se está documentando una estructura
- @class → Indica que se está documentando una clase
- @fn → Indicar que se está documentando una función

La ausencia de tags en cualquier línea escrita dentro del formato de documentación de Doxygen es asumida como parte de la descripción ampliada.

Doxygen permite documentar a pie de línea en cualquier parte del código y extender la descripción ampliada usando la siguiente expresión:

```
/**< Documentación a pie de línea */
```

Ejemplos de documentación

```
//------------------------Classes-----------------------------
/**
*      @class Test
*      @author Vicomtech member 1
*      @brief Clase de ejemplo con una funcion y una varibale
*      @version 1.0
*      @since 17/10/2013
*/
class Test
{
private:
       int var; /**< Descripción extendida de var */

public:
       Test(void);
       ~Test(void);
       int function1(int a, int b);
};
//-------------------------------------------------------------------------
```

Las funciones es preferible definirlas en el mismo sitio donde se implementa la función. Con el tag @fn Doxygen es capaz de encontrar la función a pesar de que la documentación no se las cerca de la función, al igual que pasa con el tag @class y @struct.

No obstante se asume como buena práctica el documentar la clase, estructura o método "inline" por principio de correspondencia directa entre el código y la documentación escrita con Doxygen.

```
//------------------------Functions----------------------------
/**
*      @fn int Test::function1(int a, int b)
*      @author Vicomtech member 1
*      @brief Funcion para multiplicar dos enteros
*      @version 1.0
*      @since 17/10/2013
*      @pre Dados dos numeros enteros
*      @post Devuelve la multiplicacion de ambos numeros
*      @param[in] a Numero entero
*      @param[in] b Numero entero
*      @return devuelve el resultado de multilpicar a y b
*      @note Funcion OK!!
*/
int Test::function1(int a, int b){
       return a*b;
}
//-------------------------------------------------------------------------
```

El tag @param puede usarse in la especificación [in/out] cuando todos los parámetro se asumen como entrada.

La etiqueta @note nos permite dejar apreciaciones del tipo que sean y que serán mostradas junto a la clase, estructura o función.

Las etiquetas @pre y @post constituyen un tandem que delimita las precondiciones (no tienen por qué ser sólo parámetros) y las poscondiciones que se derivan tras la ejecución del método. Sirve para definir a grandes rasgos qué necesita y qué devuelve la función planteado como predicados lógicos.

```
//-------------------------Structures-----------------------------
/**
*      @struct foo
*      @author Vicomtech member 1
*      @brief Estructura para guardar un resultado
*      @version 1.0
*      @since 15/10/201
*      @see Referencia cruzada a la clase Test
*
*      Esto es una descripción extendida para la estructura foo
*
*/
struct foo{
       int resultado; /**< Variable para guardar un valor entero */
};
//---------------------------------------------------------------------------
```

El tag @see nos permite establecer referencias cruzadas e información adicional que puede ser de interés cuando se está leyendo una parte concreta de la documentación. Cualquier clase, estructura o método mencionado en este tag se convierte en un enlace al mismo, cuando Doxygen genera la documentación.

En cualquier parte podemos extender la descripción de cualquier clase, estructura o método escribiendo fuera de las etiquetas pero dentro del bloque de documentación.

Es aconsejable establecer la versión (@version), la fecha de creación/modificación (@since) y el autor (@author). Estos tres datos permiten establecer el estado y autor de la clase, estructura o método.