UNIVERSIDAD DE ZARAGOZA

Proyecto de Fin de Carrera de Ingeniería Informática

Reconocimiento y registro 3D de objetos conocidos en una escena

Curso Académico: 2009-2010

Fecha: Septiembre 2010

Departamento de Informática e Ingeniería de Sistemas

Centro Politécnico Superior

Autor: *Félix Valdivielso Miranda*

Director: *Luis Montesano*

Codirector: *Javier Civera*

Reconocimiento y registro 3D de objetos conocidos en una escena

RESUMEN

El proyecto se inicia con la reconstrucción densa de una escena 3D a partir de imágenes en dos pasos. Con el primero de ellos se obtendrá la posición 3D de las cámaras usando la técnica conocida como Bundle Adjustment. En un segundo paso, a partir de estas localizaciones y mediante restricciones proyectivas se densificará la reconstrucción 3D de la escena. En esta primera fase del proyecto se desarrollará un visor 3D el cual nos permitirá manipular y visualizar el entorno 3D obtenido a partir de los programas mencionados previamente y que nos será de utilidad para la aplicación final.

La segunda fase del proyecto se plantea el reconocimiento de objetos a partir de imágenes. El reconocimiento se realizará basado en características salientes en la imagen. En primer lugar se creará una pequeña base de datos con imágenes de un conjunto de objetos y su reconstrucción densa. En segundo lugar, se buscará en la escena los objetos de la base de datos mediante la comparación de descriptores asociados a las características salientes. Para ello será necesario el desarrollo de una aplicación que nos permita comparar las imágenes de los diferentes objetos de nuestra base de datos con las imágenes de la escena y ver así si los objetos de la base de datos aparecen o no en la escena.

Una vez el objeto ha sido reconocido en la escena se pretende sustituir en el modelo 3D de dicha escena la reconstrucción 3D del objeto (por ejemplo, un libro) disponible en nuestra base de datos, permitiéndonos así visualizar en la escena 3D partes del libro que no se veían en las imágenes de la escena. Para ello será necesaria una tercera y última fase en el proyecto donde se deberá posicionar los modelos 3D de los objetos que disponemos en la base de datos y que aparecen en la escena.

Agradecimientos:

A mis padres por apoyarme en todo momento.

A mi hermana por estar siempre animándome.

A mis amigos por hacerme los días más amenos.

A mi abuela por preocuparse por cómo me iba el proyecto.

A mi director y codirector por su ayuda durante el proyecto.

TABLA DE CONTENIDOS

1. Introducción

A lo largo de los últimos años se ha incrementado de forma considerablemente el número de aplicaciones relacionadas con el modelado y reconstrucción 3D del entorno que nos rodea. Entre los principales motivos de este incremento encontramos que su aplicación es de gran utilidad en ciertos campos como son la robótica, medicina, arqueología, arquitectura, etc. Sin embargo, no hay que olvidar destacar como otro de los motivos de este incremento el gran avance de las tecnologías en estos últimos años, ya que los algoritmos que se utilizan hoy en día muchos de ellos ya habían sido desarrollados hace 30 años, pero debido a sus altos costes computaciones hacia que estos fueran inviables para la época. Además gracias al reciente incremento en la calidad y resolución de aparatos de visión como son las cámaras de fotos las cuales pueden llegar a tener incluso más de 20 megapixeles se pueden llegar a obtener resultados excelentes.

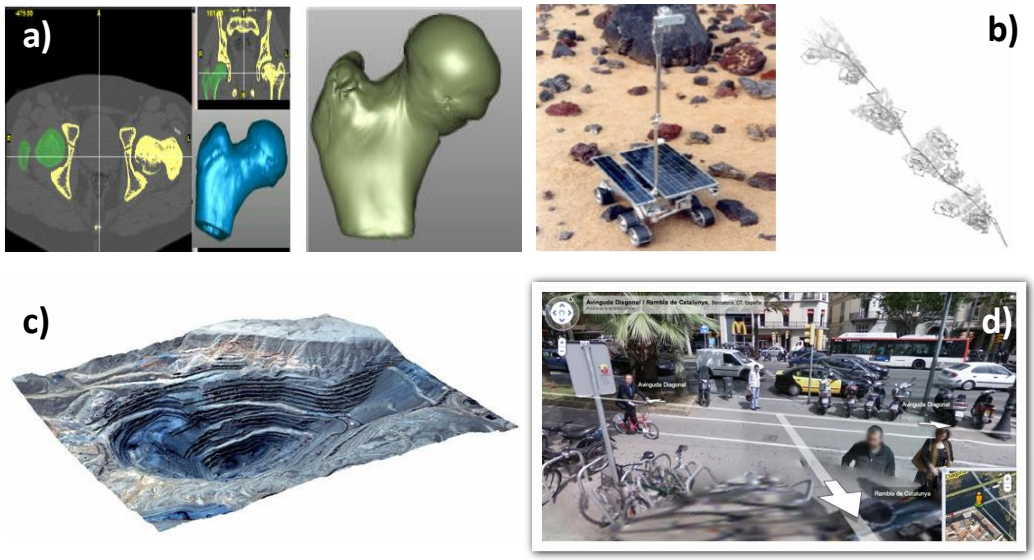


Figura 1: Campos y aplicaciones de visión por computador: (a) Medicina - Reconstrucción 3D de partes del cuerpo. (b) Robótica - Obtención de mapas en entornos desconocidos. (c) Topografía - Reconstrucción de relieves. (d) Realidad Virtual - Modelado y reconstrucción de entornos de la vida cotidiana.

Inicialmente para conseguir obtener un modelo en tres dimensiones se requería de la utilización de hardware especializado y de un gran presupuesto como es el caso de láseres. Sin embargo en los últimos años han surgido nuevas técnicas para la obtención de modelos 3D de objetos reales en los cuales no se requiere más que una cámara y un ordenador. Con la cámara se obtienen una serie de imágenes del entorno a reconstruir, de forma que introduciendo estas imágenes como elemento de entrada a ciertos algoritmos en el ordenador podemos obtener su modelo 3D.

La obtención de los modelos 3D a partir de imágenes no es automática, si no que se requiere de un proceso con varias etapas para obtenerlo, además de un cierto tiempo de cálculo dependiendo de la calidad del modelo que se desee dificultando así su obtención en un tiempo reducido. Es por ello que durante estos últimos años se ha extendido considerablemente el interés por obtener modelos 3D de calidad a partir de secuencias de video, permitiendo así calcular modelos en tiempo real.

1.1 Objetivo y Resumen del proyecto

El objetivo de este proyecto se centra en hacer uso de los conocimientos actuales sobre visión por computador para conseguir el modelado, reconocimiento y posicionamiento de objetos en un entorno 3D a partir de imágenes. Un ejemplo de lo que se busca con este proyecto sería conseguir reconocer un objeto en una escena reconstruida en 3D y poder sustituirlo por un modelo completo y con mayor nivel de detalle de ese mismo objeto.

Para ello el proyecto se divide en cuatro fases claramente diferenciadas:

1. Obtención de datasets

En esta fase se realizan fotografías de las escenas y objetos a utilizar durante el proyecto. Cada dataset está formado como mínimo por un conjunto de imágenes de una escena además de por un conjunto de imágenes de varios objetos que aparezcan en dicha escena.

2. Reconstrucción 3D

En esta segunda fase del proyecto se procede a obtener un modelo 3D de las escenas y objetos a partir de las imágenes de los distintos datasets. El proceso de reconstrucción 3D a utilizar para este proyecto se divide en dos etapas:

- Reconstrucción 3D no densa: en la que se obtienen algunos de los puntos 3D del modelo y los parámetros de las cámaras (distancia focal, coeficientes de distorsión, posición...) para las diferentes imágenes.
- Reconstrucción 3D densa: en la que a partir de los resultados obtenidos en el proceso no denso se calcula un nuevo modelo con un mayor número de puntos 3D mediante un algoritmo basado en densificación de parches.

Para poder visualizar y trabajar en un futuro con estos modelos a lo largo del proyecto se desarrollara un visor que nos permita ver los modelos 3D reconstruidos.

3. Emparejamiento

La tercera fase del proyecto consiste en buscar que objetos aparecen en la escena con cada dataset. Para ello primero haremos una comparativa de los distintos algoritmos existentes actualmente para calcular los puntos de interés (puntos más característicos de una imagen) y descriptores (información de la imagen invariante a transformaciones de rotación, translación, escala...) de una imagen. Una vez nos hayamos decantado por un algoritmo se procederá a obtener los puntos de interés tanto de las imágenes de la escena como de los objetos de los distintos datasets para posteriormente calcular los descriptores correspondientes a esos puntos. Ya con los descriptores calculados se compararan los descriptores de los diferentes objetos con los de la escena. Dependiendo del número de emparejamientos con éxito que se obtengan se podrá considerar si un objeto aparece o no en la escena. A lo largo de esta fase se desarrollará una segunda aplicación para comprobar si el algoritmo de emparejamiento cumple las expectativas y funciona correctamente.

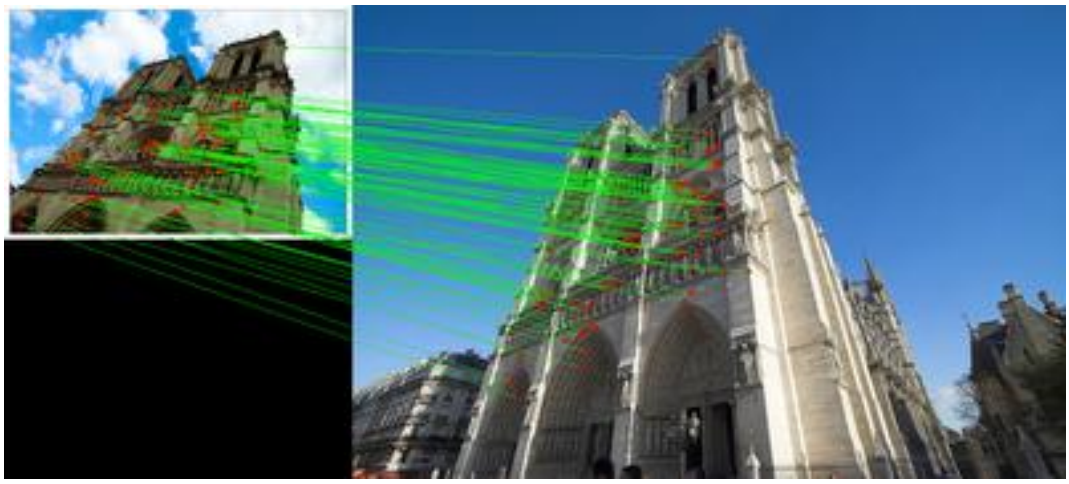
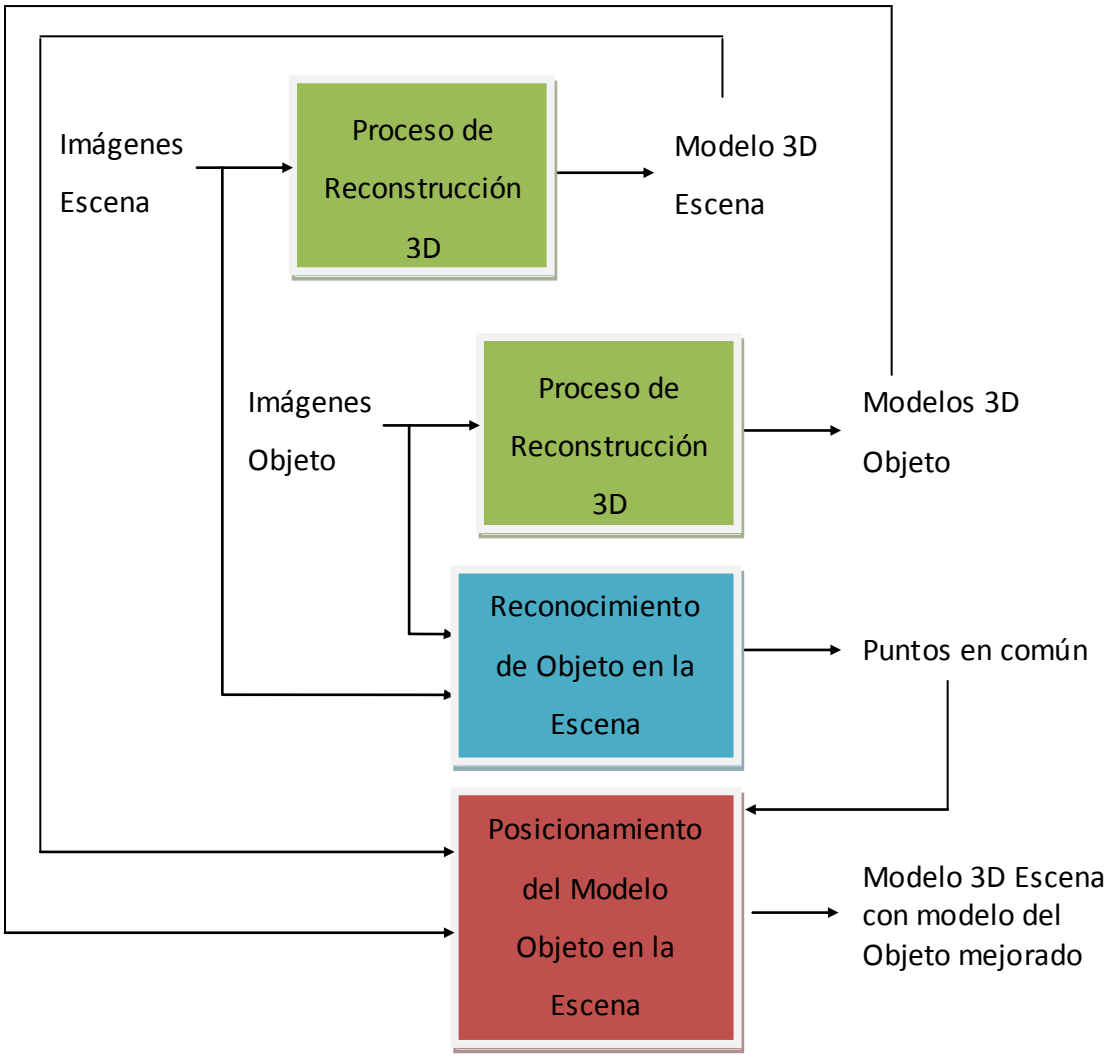


Figura 2: Ejemplo de emparejamiento entre dos imágenes. En rojo aparecen los puntos de interés encontrados y en verde las líneas que emparejan los puntos de interés de una imagen con otra comparando sus descriptores.

4. Posicionamiento

Como última parte se introducirá el modelo 3D de los objetos que aparecen en la escena en el modelo 3D de la escena. Para ello dependiendo de si se han producido suficientes emparejamientos en la fase previa entre las imágenes de la escena y los diferentes objetos procederemos a posicionar o no los objetos. En caso de haber suficientes emparejamientos se calculara con ellos la posición (rotación y translación) de la cámara de la imagen del objeto con respecto a la cámara de la escena, además de la escala de la translación obtenida y la diferencia de escala entre el modelo 3D de la escena y el objeto. De esta forma con los parámetros de las cámaras obtenidos en el proceso de reconstrucción 3D y los ahora calculados se procederá a posicionar el modelo 3D del objeto en el modelo 3D de la escena.



Esquema básico del proyecto para un dataset.

1.2 Aplicaciones del Proyecto

A continuación se procede a poner algunos ejemplos en los que se podría aplicar algunas funcionalidades desarrolladas en este proyecto.

1.2.1 Realidad Aumentada

La realidad virtual es un término referido a la visión directa o indirecta de un entorno físico real cuyos elementos se combinan con elementos virtuales para la creación de una realidad mixta a tiempo real.

Un ejemplo de aplicación de este proyecto a la realidad aumentada seria el reconocimiento de objetos en tiempo real. Gracias a ello podríamos recorrer una galería o museo con un dispositivo que fuera visualizando las imágenes o cuadros para en tiempo real consultar una base de datos y mostrarnos en la pantalla del dispositivo información sobre lo que se está visualizando. Con esta tecnología sería posible y muy cómodo para una persona por ejemplo que viaja a otro país conocer detalles de los monumentos, locales y sitios de dicho lugar simplemente utilizando este dispositivo.

Figura 3: Ejemplo de reconocimiento de imágenes en tiempo real.

Otra posible aplicación a realidad aumentada seria por como tutorial para ciertos dispositivos, tras reconocer por ejemplo un osciloscopio se podría señalar mediante la pantalla del dispositivo partes del osciloscopio y añadir instrucciones permitiendo así al usuario aprender sobre este dispositivo de una forma sencilla y rápida. Este ejemplo podría ser muy útil para facilitar la realización de ciertas prácticas en la universidad en las que se requiere de la utilización de elementos o maquinaria nueva para el estudiante, pudiendo así guiar la practica con el dispositivo de realidad virtual haciéndole más fácil adaptarse al entorno.

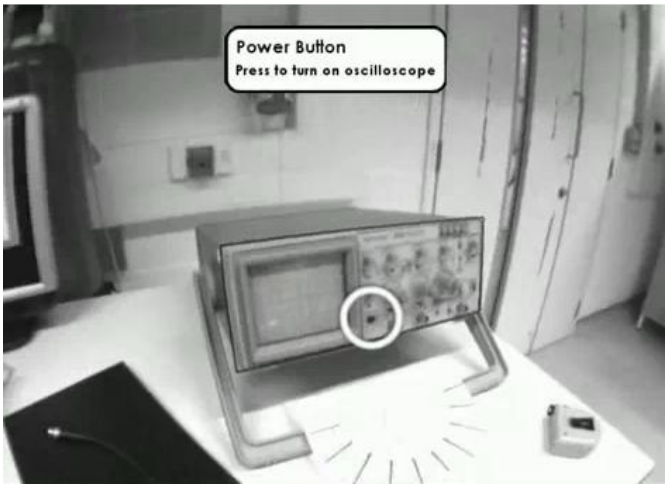


Figura 4: Ejemplo de tutorial para un osciloscopio mediante realidad aumentada.

1.2.2 Robótica

El mundo de la robótica en la industria está especialmente acotado por la precisión y la repetición haciendo que por ejemplo un brazo robótico realice siempre el mismo recorrido y los mismos movimientos. Sin embargo la tecnología está en constante evolución y es muy frecuente que cada cierto tiempo sea necesario modificar los movimientos que realiza un robot debido a modificaciones en el producto de fabricación que conllevan a un cambio en su entorno, llevando con ello un enorme coste en tiempo y dinero hasta que se consigue que el robot realice de nuevo su trabajo para ese nuevo entorno. Seria por tanto interesante que el robot pudiera

mediante varias cámaras reconocer objetos y saber su posición en el entorno de trabajo pudiendo así interactuar con ellos sin necesidad de programar previamente cada movimiento que ha de realizar.

Figura 5: Ejemplo de un brazo robótico en una cadena de montaje.

Un ejemplo claro de aplicación en el mundo de la industria seria el proceso de pintado en una cadena de montaje de coches, en donde para programar los movimientos que ha de realizar el brazo robótico para pintar un vehículo se utiliza a una persona que desplaza el brazo por las diferentes partes de la carrocería para así conocer las coordenadas exactas. Si se tuviera por ejemplo una reconstrucción 3D de la carrocería a pintar por el brazo robótico y se supiera su localización exacta en el entorno se podría pintar la carrocería sin necesidad de programar los movimientos del brazo previamente.

2. Obtención de Datasets

Para la realización de este proyecto se ha decidido utilizar dos datasets:

El primer dataset consta de una escena exterior en la que los objetos que aparecen poseen escasa textura haciéndolos más difíciles de identificar. Para ello se decidió utilizar como escena del primer dataset la cafetería del edificio Ada Byron del C.P.S. y como objetos las sillas y mesas de la cafetería. Como se puede observar en la figura 5 las sillas y las mesas que aparecen apenas tienen textura al ser prácticamente en su totalidad de un único color sin elementos identificables.

Figura 6: Fotografía realizada a la escena del primer dataset.

El segundo dataset consta de una escena interior con objetos con una textura más característica y por lo tanto más fácilmente identificable. En este caso se ha optado por utilizar como escena interior una mesa con varios objetos repartidos sobre ella, entre algunos de los objetos a utilizar en este dataset encontramos un peluche, un libro, un escarabajo, etc.

Figura 7: Fotografía realizada a la escena del segundo dataset.

A la hora de obtener las imágenes de los distintos datasets se ha optado por realizar las fotografías de la escena desde un único lado de esta para que al obtener posteriormente el modelo 3D existan partes de los objetos de la escena sin reconstruir debido a que no se tomaron fotos de esas partes. En el caso de los objetos por separado es distinto ya que lo que queremos es disponer del modelo 3D completo del objeto para posteriormente posicionarlo, por lo que tomaremos fotografías desde todos los ángulos.

3. Reconstrucción 3D a partir de imágenes

Como se ha mencionado en la introducción para obtener los modelos 3D de un entorno a partir de un conjunto de imágenes de este se ha decidido realizar dos etapas: una en la que se obtienen algunos puntos del modelo y los parámetros de las cámaras para cada fotografía (reconstrucción 3D no densa) y una segunda para a partir de los datos obtenidos en la etapa previa obtener un modelo 3D con más puntos (reconstrucción 3D densa).

Antes de proceder con el siguiente apartado se recomienda consultar el anexo 1 sobre conceptos básicos de visión por computador para facilitar la comprensión de las ecuaciones. Los métodos descritos a continuación sobre reconstrucción 3D no densa y densa son métodos básicos de visión por computador, pero hay que tener en consideración que no son la única forma de realizarlos.

3.1. Reconstrucción 3D no densa

Esta primera fase de reconstrucción 3D se centra en la obtención de los parámetros de la cámara para cada fotografía como son su posición, distancia focal, coeficientes de distorsión, etc. Además, en esta fase se obtiene la posición 3D de algunos de los puntos de la escena fotografiada.

Para ello inicialmente se calculan los puntos de interés de cada imagen de la escena que queremos obtener el modelo 3D, siendo $x_{i,j} = [x\ y\ 1]$ el punto de interés j de la imagen i. Una vez se dispone de los puntos de interés para cada imagen se procede a calcular una matriz fundamental que cumple la siguiente propiedad

$$\boldsymbol{x_{i,j}}^{\boldsymbol{'T}} * \boldsymbol{F} * \boldsymbol{x_{m,n}} = \mathbf{0}$$

Siendo F la matriz fundamental, $x'_{i,j} = [x\ y\ 1]$ el punto de interés j de la imagen i y $x_{m,n} = [x\ y\ 1]^T$ el punto de interés n de la imagen m emparejado con el punto x'. Así pues según la ecuación descrita la matriz fundamental relaciona las correspondencias entre pixeles de dos imágenes.

Si calculamos F y conocemos los parámetros intrínsecos de las cámaras (distancia focal, coeficientes de distorsión, etc.) podemos obtener la rotación y translación de una cámara con respecto a otra ya que:

$$\boldsymbol{K}^{'\boldsymbol{T}} * \boldsymbol{F} * \boldsymbol{K} = \boldsymbol{t} \times \boldsymbol{R}$$

Siendo K y K' las matrices de parámetros intrínsecos de las cámaras, F la matriz fundamental y t y R el vector de translación y la matriz de rotación de la cámara primera a la segunda. Es decir, siendo P y P' las matrices de proyección de cada camara, e I una matriz de identidad 3x3 se cumple lo siguiente:

$$\boldsymbol{P} = [\boldsymbol{I}\ |\mathbf{0}] \text{ y } \boldsymbol{P}' = [\boldsymbol{R}\ |\boldsymbol{t}]$$

Para calcular la matriz fundamental se necesita de un mínimo de 7 correspondencias:

$$\boldsymbol{x'^{T}} * \boldsymbol{F} * \boldsymbol{x} = \mathbf{0}$$

$$\boldsymbol{x'xf_{11} + x'yf_{12} + y'xf_{13} + y'xf_{21} + y'yf_{22} + y'f_{23} + xf_{31} + yf_{32} + f_{33}} = \mathbf{0}$$

Separando términos conocidos de los términos no conocidos:

$$[\boldsymbol{x'x, x'y, x', y'x, y'y, y', x, y, 1}] * [\boldsymbol{f_{11}, f_{12}, f_{13}, f_{21}, f_{22}, f_{23}, f_{31}, f_{32}, f_{33}}]^T = \mathbf{0}$$

$$\begin{bmatrix} \boldsymbol{x'_1x_1} & \boldsymbol{x'_1y_1} & \boldsymbol{x'_1} & \boldsymbol{y'_1x_1} & \boldsymbol{y'_1y_1} & \boldsymbol{y'_1} & \boldsymbol{x_1} & \boldsymbol{y_1} & \mathbf{1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \boldsymbol{x'_nx_n} & \boldsymbol{x'_ny_n} & \boldsymbol{x'_n} & \boldsymbol{y'_nx_n} & \boldsymbol{y'_ny_n} & \boldsymbol{y'_n} & \boldsymbol{x_n} & \boldsymbol{y_n} & \mathbf{1} \end{bmatrix} * \boldsymbol{f} = \mathbf{0}$$

El algoritmo básico seguido en este proyecto para la reconstrucción no densa consta de las siguientes fases:

1. Encontrar N vías (conjunto de puntos de interés que conectan múltiples imágenes) $T = \{T_1, T_2, \dots, T_N\}$
    a. Para cada par de imágenes $\{Q_i, Q_j\}$:
        - Detectar los puntos de interés en Q_i y Q_j.
        - Emparejar los puntos de interés y posteriormente utilizar un filtrado RANSAC para eliminar emparejamientos erróneos.
    b. Emparejar los puntos de interés a través de múltiples imágenes, construyendo vías.
2. Estimar $\{P_1 \dots P_N\}$ y la posición 3D de para cada vía $\{X_1 \dots X_N\}$
    a. Seleccionar un par de imágenes $\{Q_{1'}, Q_{2'}\}$ (bien condicionadas, es decir que tengan un gran número de emparejamientos y no compartan un ángulo de visión muy similar)
    b. Siendo $T_{1'2'}$ las vías que contienen puntos en común de ambas imágenes, estimar $K_{1'}$ y $K_{2'}$, calcular $\{P_{1'}, P_{2'}\}$ y la posición 3D de $T_{1'2'}$ a partir de la matriz fundamental. Minimizar los errores de proyección de forma no lineal.
    c. Incrementalmente añadir una nueva cámara P_K al sistema, estimar los parámetros de la cámara y optimizar el sistema de forma no lineal.
    d. Repetir el paso c hasta que todas las cámaras hayan sido estimadas.

Para un mayor nivel de detalle del proceso seguido por este algoritmo consultar la sección 9 de anexos.

En nuestro caso, se ha decidido utilizar Bundler, un software desarrollado por Noah Snavely [1] para obtener los parámetros de la cámara. La aplicación Bundler es de código abierto (escrita en C), actualmente se encuentra en su tercera versión y se considera una aplicación bastante robusta y potente. Un ejemplo de ello es su utilización para el proyecto Photo Tourism, en el cual a partir de una larga colección de imágenes de usuarios de internet se obtienen los modelos 3D de monumentos tan reconocidos como son la catedral de Notre Dame o el coliseo de Roma (Ver la figura 8).

Figura 8: Ejemplo de la aplicación Bundler para el proyecto Photo Tourism. [12]

3.1.1. Explicación del proceso

Para empezar con la fase de reconstrucción no densa se procedió a instalar Bundler junto a sus dependencias. Pese a que Bundler dispone de un manual de instalación, fue necesario dedicarle cierto tiempo a la instalación y configuración de este debido a la gran cantidad de dependencias con otras librerías.

Ya con Bundler instalado se procedió a introducir las imágenes de los distintos datasets y comprobar si mediante estas era posible obtener los parámetros de las cámaras correctamente.

Para la obtención de resultados coherentes hay que tener en cuenta que Bundler consulta inicialmente un fichero donde almacena una lista con información CCD de las cámaras más conocidas del mercado. Básicamente lo que hace Bundler es buscar en la información EXIF de la fotografía el modelo de la cámara para posteriormente consultar en su lista dicho modelo y así saber la distancia focal de la

fotografía. Sin embargo, entre las cámaras de dicha lista no aparecía nuestra cámara por lo que fue necesario añadir la siguiente línea al fichero que contiene la lista.

```
"OLYMPUS IMAGING CORP. SP310"      => 7.176, # 1/1.8"
"OLYMPUS IMAGING CORP.   SP510UZ"  => 5.75,   # 1/2.5"
"OLYMPUS IMAGING CORP.   u9000,S9000" => 6.13, # 1/2.33"
"OLYMPUS IMAGING CORP.   SP550UZ" => 5.76, # 1/2.5"
"OLYMPUS IMAGING CORP.   uD600,S600" => 5.75, # 1/2.5"
```

Donde u9000,S9000 es el modelo de nuestra cámara según la información EXIF de la fotografía, 6.13 es el ancho CCD y 1/2.33 es la relación alto/ancho CCD.

Una ejecución del programa Bundler costo en nuestro caso entre 20 y 30 minutos para un conjunto de 40-50 fotografías en un ordenador convencional. Durante la ejecución pudimos apreciar que aproximadamente la mitad del tiempo de ejecución es empleado para el cálculo de los puntos de interés y descriptores de cada imagen, mientras que el resto del tiempo es empleado en realizar emparejamientos entre las imágenes y obtener los parámetros de las cámaras y algunos de los puntos 3D del modelo.

3.2. Reconstrucción 3D densa

En esta segunda fase de la reconstrucción 3D se obtiene a partir de los resultados previos un modelo más detallado de la escena a representar.

El algoritmo a utilizar para esta fase consta de tres etapas: emparejamiento, expansión y filtrado. A lo largo de la primera fase se calculan los puntos de interés de las imágenes para posteriormente realizar un proceso de emparejamiento con estos para obtener un conjunto de parches asociados con las regiones más destacadas de las imágenes. La segunda fase se encarga de expandir los emparejamientos iníciales a los pixeles cercanos y así obtener un conjunto denso de parches. Como última fase se utiliza la limitación de visibilidad de las cámaras para eliminar emparejamientos incorrectos.

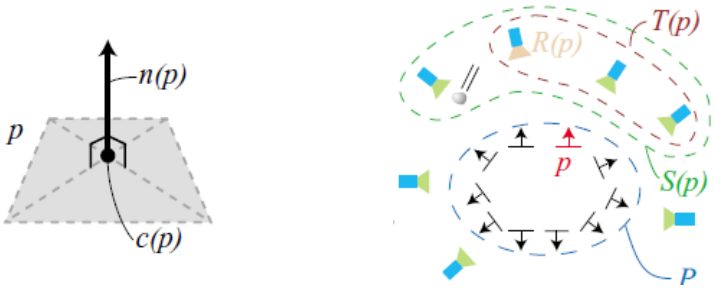


Figura 9: (izq) Ejemplo de parche, donde p indica un parche, n(p) es la normal y c(p) es el centro del parche. (der) Ejemplo de escena con parches donde S(p) es el conjunto de cámaras donde el parche p debería ser visible, T(p) es el conjunto de cámaras real donde p es visible y R(p) es la cámara con respecto a la cual se obtiene la orientación y la extensión de p. [2]

1) Emparejamiento

Durante esta primera fase se calculan inicialmente los puntos de interés de las imágenes mediante el método de diferencia de gaussianas (DoG). Una vez se dispone de los puntos de interés para todas las imágenes se procede para cada punto de interés f en la imagen I_i a encontrar en el resto de imágenes el conjunto F de puntos

de interés f' que se emparejan con f. El emparejamiento (f, f') ha de cumplir que la distancia de dichos puntos a sus correspondientes líneas epipolares no supere un número determinado de pixeles.

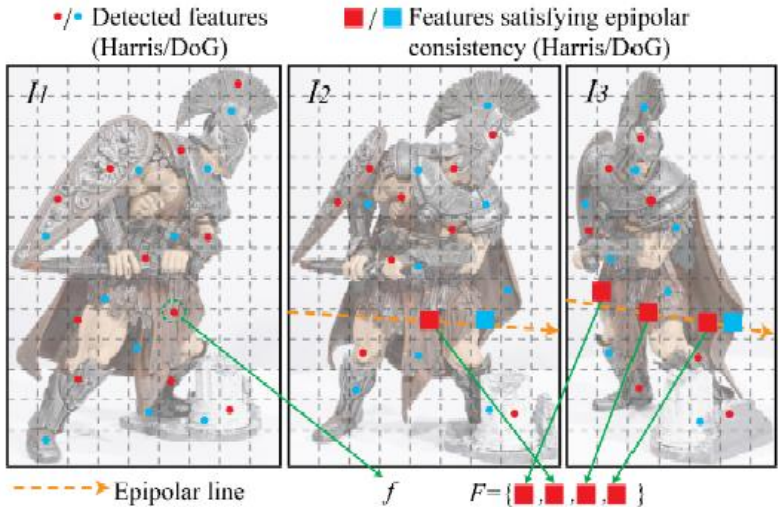


Figura 10: Ejemplo de emparejamiento basándonos en las líneas epipolares. [2]

2) Expansión

Para facilitar el acceso a los parches vecinos se crea una malla con celdas de tamaño $\beta_1 x\, \beta_1$ para cada imagen. Para saber a qué celda pertenece un parche simplemente se proyecta este en la imagen. Entonces cada celda representa el conjunto de parches que se proyectan sobre ella.

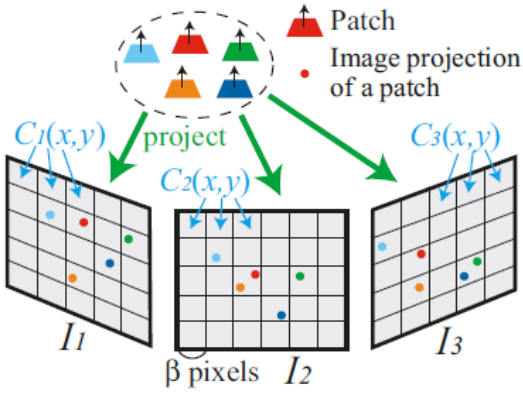


Figura 11: Ejemplo de malla para tres imágenes que contienen parches en común. [2]

El objetivo del algoritmo de expansión es reconstruir al menos un parche para cada celda de las imágenes, repetimos el proceso tomando parches que ya tenemos y generando nuevos en las celdas vecinas vacías.

El algoritmo de expansión sigue el siguiente procedimiento:

Entrada: Parches P procedentes el proceso de emparejamiento

Salida: Conjunto de parches expandidos

Mientras que P no esté vacio

    Seleccionar y eliminar un parche p de P;

    Para cada celda de la imagen $C_i(x,y)$ que contenga a p

        Seleccionar un conjunto C de celdas para expandir;

        Para cada celda $C_i(x',y')$ en C

            Crear un parche candidato p';

            Calcular los parámetros del parche candidato p';

            Si el parche candidato es visible en más de γ

            imágenes se añade p' a P;

3) Filtrado

En esta última fase se aplican tres filtros para eliminar parches erróneos (outliers). Los dos primeros filtros se basan en aspectos de visualización para detectar outliers, mientras que el tercer filtro se centra en la vecindad del parche para detectar si es un outlier.

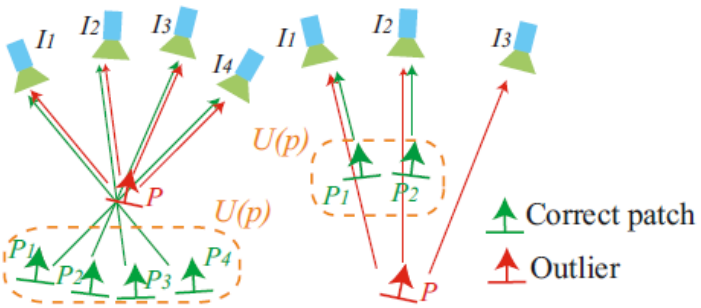


Figura 12: Ejemplo del primer proceso de filtrado de parches. U(p) denota un conjunto de parches que son inconsistentes visualmente con la información de P, por lo que se identifica a P como un outlier. [2]

Para realizar el proceso de reconstrucción 3D densa se ha decidido utilizar un segundo programa llamado PMVS, un software desarrollado por Yasutaka Furukawa [2], el cual también es de código abierto (escrito en C) . Este programa toma inicialmente un conjunto de imágenes, además de los parámetros de las cámaras para reconstruir el modelo 3D del objeto o la escena que aparece en las imágenes mediante el método descrito previamente. PMVS solo reconstruye estructuras rígidas, es decir ignora los objetos no rígidos como podría ser el caso de peatones caminando enfrente de un edificio. Como salida del programa obtenemos un conjunto de puntos orientados (sus coordenadas, la normal a la superficie y el color) en lugar de modelos poligonales.

Figura 13: Reconstrucción 3D del coliseo de Roma con PMVS. [13]

3.2.1. Explicación del proceso

Al igual que con Bundler pese a que PMVS también dispone de un manual de instalación, fue necesario dedicarle cierto tiempo a la instalación de PMVS para hacerlo funcionar correctamente debido a su gran cantidad de dependencias.

Tras instalar correctamente PMVS en el sistema se procedió a obtener el modelo en 3D del entorno que aparece en las fotografías utilizadas previamente en el Bundler, para ello fue necesario preparar los datos de entrada antes de la ejecución.

PMVS requiere como datos de entrada una matriz de proyección para cada cámara, además de las imágenes correspondientes a cada cámara y un fichero de configuración del proceso de reconstrucción.

Bundler dispone de un ejecutable, el cual nos permite convertir los parámetros que obtuvimos con este de cada cámara (rotación, translación, distancia focal etc.) a su correspondiente matriz de proyección. De esta forma, nos facilita de gran medida el proceso de introducción de los datos de las cámaras a la aplicación PMVS.

Una vez ya disponíamos de los datos de entrada se procedió a ejecutar PMVS para obtener los modelos 3D. El tiempo de ejecución de esta aplicación es medianamente elevado llevando en nuestro caso entre 20 y 40 minutos en un ordenador convencional. Hay que tener en cuenta que dependiendo del número de fotografías empleadas para la reconstrucción, el tamaño de estas, los parámetros empleados en el fichero de configuración etc. el tiempo se incrementara o reducirá considerablemente además de mejorar o empeorar la calidad del modelo obtenido.

3.3. Visor 3D

Para poder comprobar que los resultados obtenidos a partir de Bundler y de PMVS son correctos y que por lo tanto no se cometieron errores en la introducción de los parámetros correspondientes a su ejecución es necesaria la realización de un visor 3D. El objetivo de este visor 3D es ser capaz de leer los ficheros proporcionados por ambos programas para posteriormente mostrar estos datos en un entorno 3D. En el caso del Bundler, deberá mostrar en un entorno 3D las posiciones de las cámaras reconocidas además de los puntos de la escena u objeto. En el caso del PMVS, se deberá mostrar la reconstrucción 3D de la escena u objeto.

Para mejorar la interacción del usuario con el visor 3D se ha decidido que este pueda moverse en torno al modelo 3D con ayuda exclusivamente del ratón, permitiéndole así movimientos de rotación, translación y zoom.

Para el desarrollo de este visor se ha decidido utilizar Qt, una biblioteca multiplataforma para desarrollar interfaces graficas. Su uso actualmente está muy extendido, pudiendo encontrar aplicaciones como Google Earth, Adobe Photoshop Album, Skype... desarrolladas con esta biblioteca. Qt utiliza como lenguaje de programación C++ de forma nativa, además de permitir utilizar otros lenguajes de programación mediante bindings. Qt soporta otras librerías como son SQL, OpenGL, XML dándole así una gran versatilidad.

En cuanto al entorno de programación se decidió utilizar Microsoft Visual Studio 2008 por su facilidad de manejo y por su compatibilidad para trabajar con la biblioteca Qt.

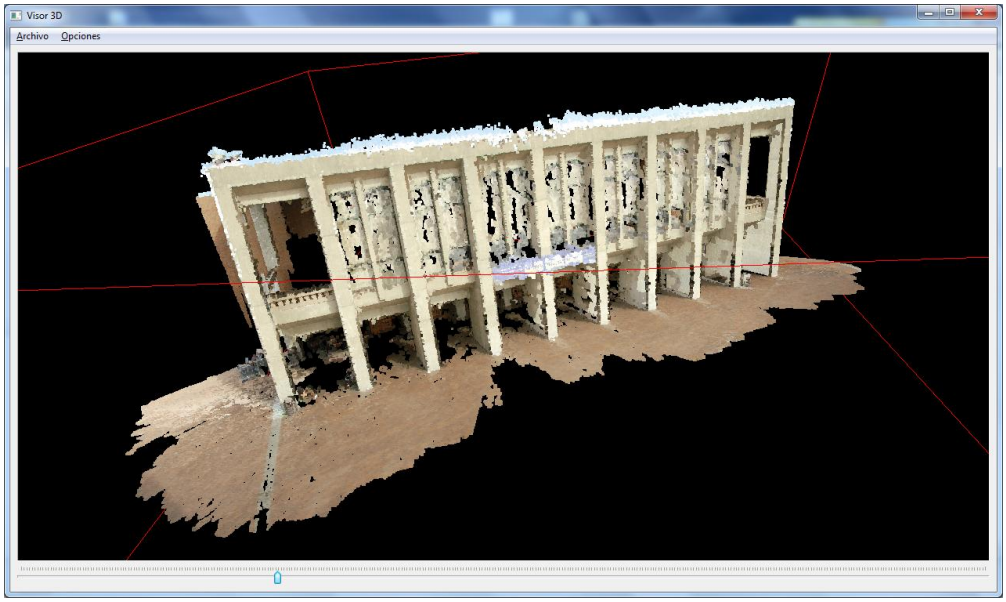


Figura 14: Imagen del Visor 3D.

3.4. Resultados

En este apartado se muestran, utilizando el visor desarrollado, los resultados obtenidos con Bundler y PMVS tras proporcionar a estos las imágenes correspondientes al primer dataset. Para ver los resultados obtenidos con el resto de entornos y objetos fotografiados acudir a la sección Datasets en Anexos.

Figura 15: Ejemplo de las fotos proporcionadas a Bundler y PMVS.

Uno de los ficheros que Bundler devuelve como salida contiene un conjunto de puntos con su localización en un entorno 3D y su correspondiente color. En este conjunto de puntos se encuentran algunos puntos pertenecientes a la escena reconstruida, mientras que otros corresponden a la posición de las cámaras. En la Figura 16 se puede observar de color rojo, verde y amarillo algunas de las posiciones de las cámaras además de algunos de los puntos de la cafetería.

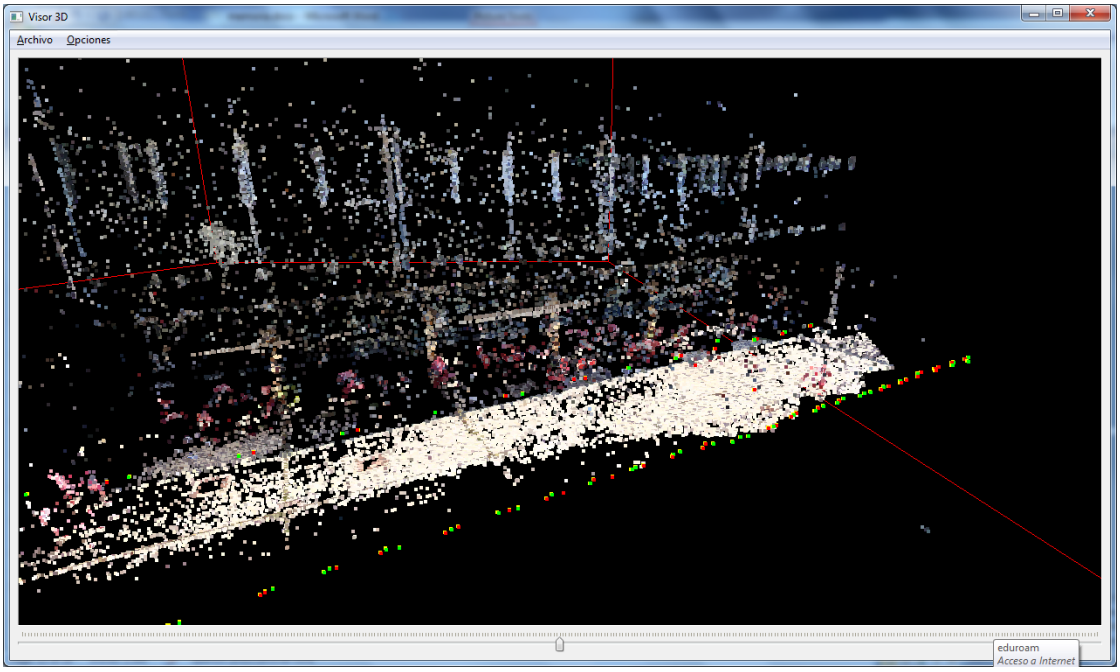


Figura 16: Imagen de la posición de las cámaras en la cafetería del C.P.S.

En el caso de PMVS se reconstruye la escena con las imágenes de las cuales se conoce su localización gracias al proceso previo con Bundler. En la figura 17 se puede visualizar la reconstrucción de parte de la cafetería. La reconstrucción no es perfecta viéndose algunos espacios vacios, esto puede ser debido a ángulos muertos, a que las fotografías proporcionaban poca información de esas zonas, etc. Para mejorar el modelo obtenido se podría aplicar posteriormente a estos resultados filtros, métodos de expansión, etc. pero debido a que la finalidad de este proyecto es el reconocimiento y posicionamiento de objetos y una mejora de la calidad de los modelos obtenidos no influiría para nada en los objetivos se ha decidido trabajar directamente con los modelos proporcionados por PMVS.

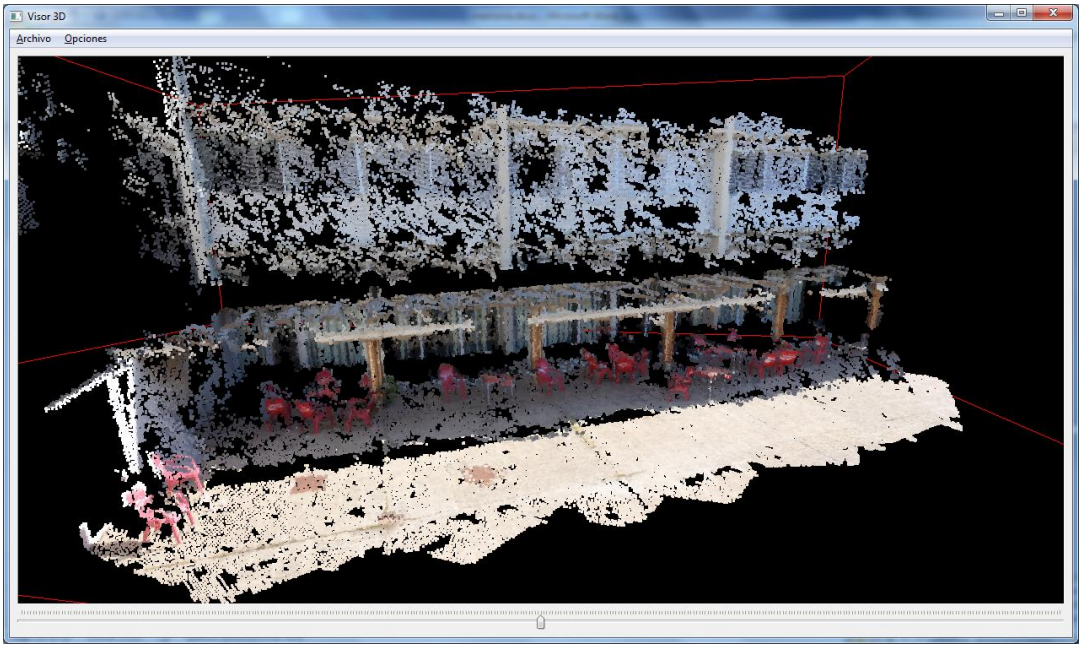


Figura 17: Imagen de la cafetería tras la reconstrucción densa.

4. Correspondencias entre puntos

Una vez se dispone de una base de datos de imágenes y modelos 3D con los que poder trabajar se requiere de un algoritmo para la obtención de descriptores locales en imágenes. Básicamente lo que hacen estos algoritmos es recopilar información invariante de pequeñas regiones de la imagen que considera de interés. Gracias a ello podemos comparar imágenes en las que aparecen los mismos objetos o escenas y saber así su grado de similitud.

Existen multitud de algoritmos que nos permiten obtener esta información, cada uno con sus ventajas e inconveniente. A continuación procedemos a explicar y comparar algunos de los algoritmos que se tuvieron en cuenta para este proyecto.

4.1 SIFT

SIFT (Scale-invariant feature transform) es un algoritmo de visión por computador para detectar y describir características locales de imágenes. El algoritmo fue publicado por David Lowe en 1999.

SIFT puede de forma robusta identificar objetos desordenados o incluso parcialmente ocultos, esto es debido a que los descriptores proporcionados por SIFT son invariantes a escala, orientación, translación y parcialmente invariante a cambios de iluminación. Los descriptores calculados por SIFT guardan gran similitud con las neuronas del cortex inferior temporal, las cuales son usadas para reconocimiento de objetos en la visión humana permitiéndonos identificar rápidamente objetos que conocemos independientemente de su ángulo de visión, color, etc.

Cada uno de los datos extraído por SIFT se considera una característica de la imagen y se describe mediante su posición, escala, orientación y su vector descriptivo (normalmente de tamaño 128).

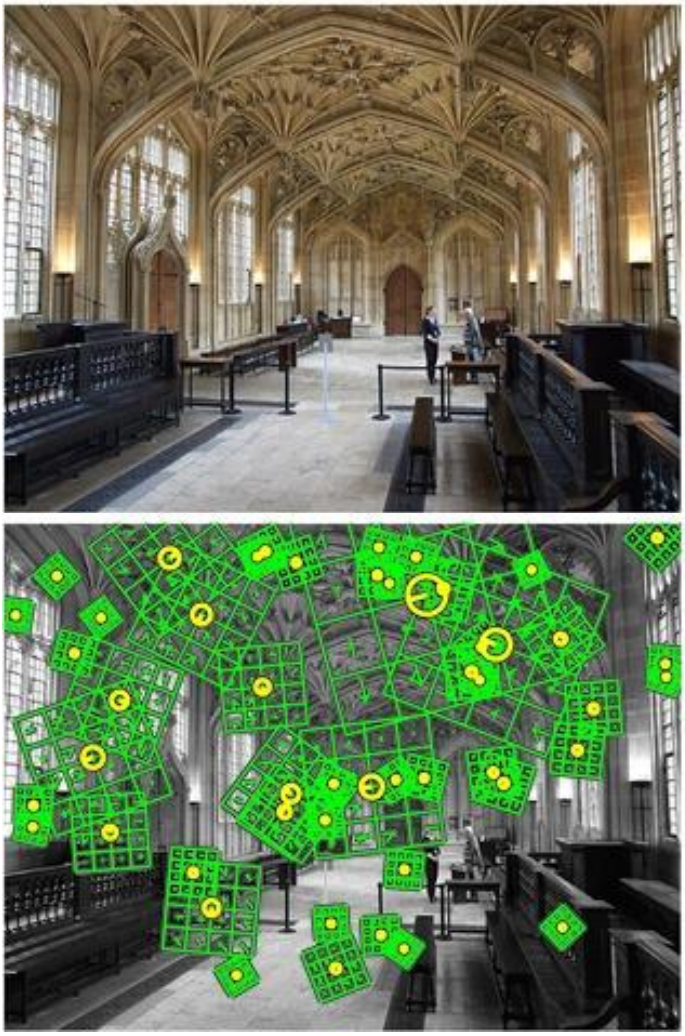


Figura 18: Ejemplo del resultado de aplicar SIFT a una imagen. [14]

4.1.1 Calculo de los descriptores

SIFT aplica 4 fases claramente diferenciadas para la extracción de los descriptores de una imagen:

1. **Detección de máximos y mínimos de la escala**

En esta fase se intenta identificar aquellas localizaciones y escalas que son identificables de diferentes vistas de un mismo objeto. Para ello se hace uso de una función del espacio de la escala. Además se ha demostrado con criterios razonables que esta se debe basar en la función de Gauss. El espacio de la escala está definido por la función:

$$L(x,y,\sigma) = G(x,y,\sigma) * I(x,y)$$

Donde * es el operador de convolución, $G(x,y,\sigma)$ es una variable de escala Gaussiana e *I(x, y)* es la imagen de entrada.

A partir de aquí se pueden aplicar varias técnicas para el cálculo de la localización de puntos de interés en el espacio de la escala. El método de diferencias Gaussianas es una de estas técnicas. En este método se calcula la diferencia entre dos imágenes, una con escala k veces mayor que la otra. Con este método se pretende obtener los extremos del espacio de la escala.

$$D(x,y,\sigma) = L(x,y,k\sigma) - L(x,y,\sigma)$$

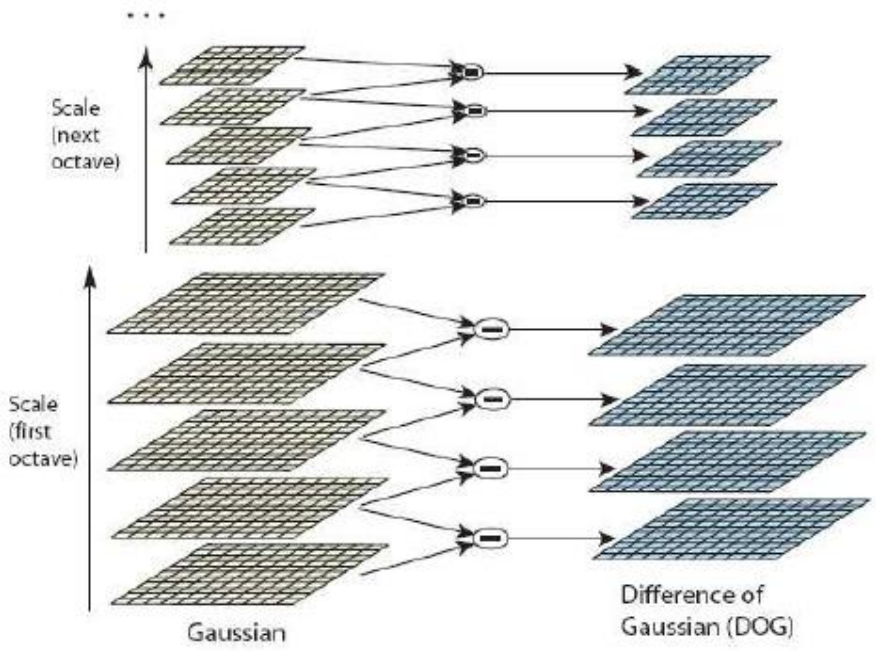


Figura 19: Representación del proceso que sigue cada octava en el espacio de la escala. [10]

Para detectar los máximos y mínimos locales de $D(x,y,\sigma)$ se compara cada punto con sus 8 vecinos de la imagen actual y sus 9 vecinos de las imágenes superior e inferior. Si dicho valor es el menor o mayor de todos los puntos entonces el punto es un extremo. De este modo se seleccionan únicamente los puntos más estables. Hay que mencionar que es posible encontrar varias escalas validas para una misma posición (x,y) de la imagen, por lo que al final será posible disponer de varios descriptores para un mismo punto.

2. **<u>Localización de puntos de interés</u>**

En esta etapa se intenta eliminar algunos puntos de la lista de puntos clave obtenidos previamente. Para ello se buscan aquellos que tienen un bajo contraste o que están pobremente localizados en una esquina. Para ello se hace uso de la siguiente función:

$$\boldsymbol{z = \frac{-\partial^2 D^{-1}}{\ddot{\sigma} x^2}\frac{\partial D}{\ddot{\sigma} x}}$$

Si el valor de la función z es menor de un determinado valor o threshold entonces ese punto es excluido.

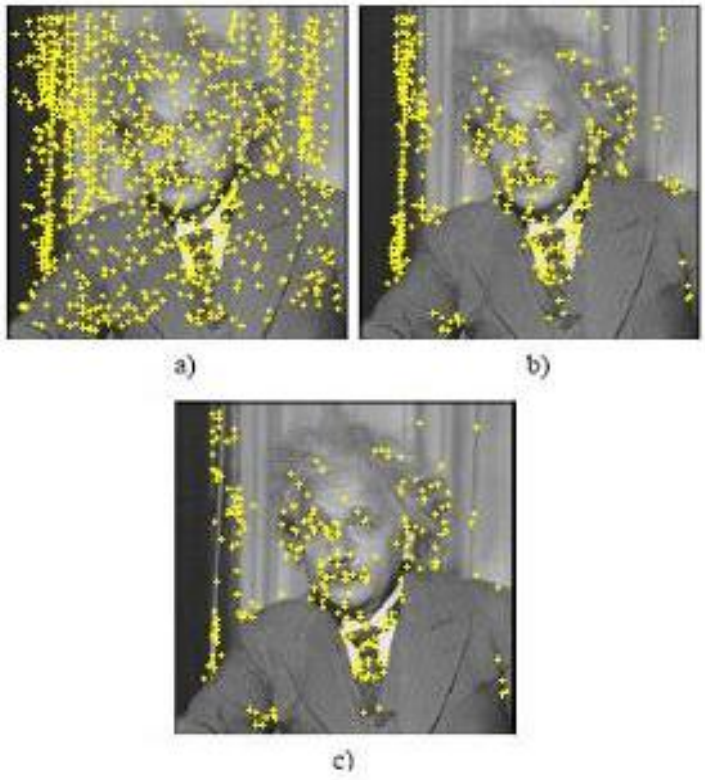


Figura 20: a) Puntos de interés detectados tras la primera fase. b) Puntos restantes tras quitar los puntos con bajo contraste. c) Puntos restantes tras quitar los que se encuentran pobremente localizados en alguna esquina. [10]

3. **Asignación de la orientación**

El objetivo de esta fase es asignar una orientación consistente a los puntos de interés basándonos en las propiedades locales de las imágenes. Los descriptores de los puntos de interés pueden ser representados relativamente a esta orientación, consiguiendo que sean invariantes a rotación. Los pasos necesarios para obtener la orientación son:

1) Usar la escala obtenida previamente de los puntos de interés para seleccionar el Gaussiano que suaviza la imagen L
2) Calcular el gradiente m

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$

3) Calcular la orientación θ

$$\Theta(x,y) = tan^{-1}\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}$$

4) Hacer un histograma de orientaciones de los gradientes de orientación de los puntos clave
5) Localizar el pico más alto del histograma. Usar este pico y cualquier otro pico local con un 80% del valor de este pico para crear un punto clave con esa orientación.
6) Algunos puntos se asignaran a varias orientaciones.
7) Ajustar una parábola para los 3 valores del histograma más próximos a cada pico para interpolar la posición de los picos.

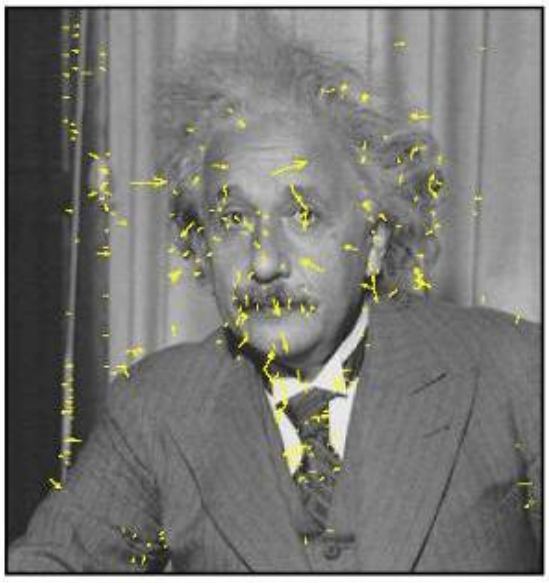


Figura 21: Puntos Clave extraídos, las flechas indican escala y orientación. [10]

4. **Descriptor del punto de interés**

Los datos del gradiente local utilizado anteriormente también se utilizan para calcular los descriptores de los puntos clave. Para asegurar la invarianza a la orientación la información del gradiente se rota para alinearla con la orientación del punto clave. A continuación se pondera por una gaussiana con una variación de 1.5 * escala del punto clave. Estos datos se utilizan para crear un conjunto de histogramas sobre una ventana centrada en el punto clave.

Los descriptores de los puntos clave normalmente usan un conjunto de 16 histogramas, alineados en una rejilla de 4x4, cada una con 8 celdas de orientación, una para cada una de las principales direcciones de la brújula y una para cada uno de los puntos medios de estas direcciones. Esto resulta en un vector de características con 128 elementos (8 orientaciones * vector de 4x4).

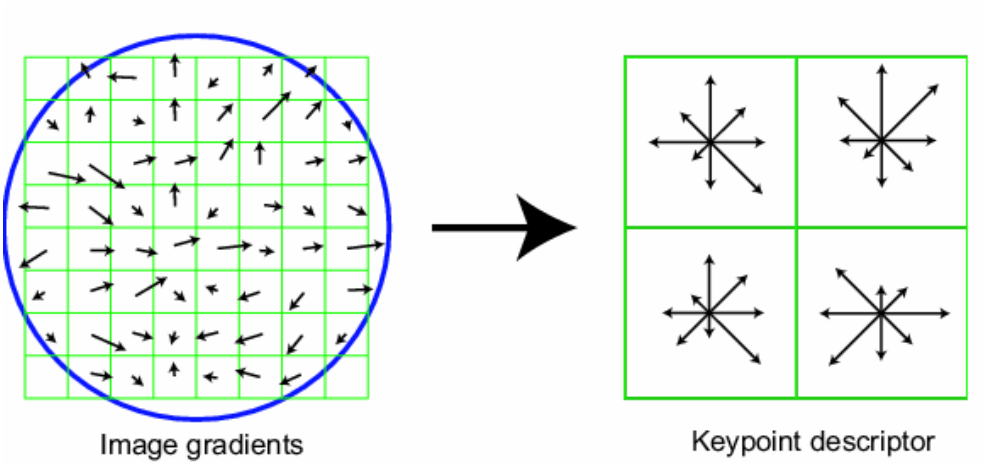


Figura 22: Ejemplo de gradientes en una imagen con SIFT. [10]

4.2 SURF

SURF (Speeded Up Robust Features) es un robusto algoritmo para calculo de puntos de interés invariantes y descriptores en imágenes. Este algoritmo fue presentado por Herbert Bay en 2006. Dicho algoritmo fue inspirado inicialmente por los descriptores SIFT. Según su autor tiene como ventaja sobre SIFT su rapidez en el cálculo además de ser más robusto con las transformaciones de las imágenes.

SURF en primer lugar detecta los posibles puntos de interés y su localización en la imagen. Una vez obtenidos los puntos se define un vector (normalmente de 64 elementos) para cada punto donde queda representada su vecindad.

4.2.1 Calculo de descriptores

SURF utiliza imágenes integrales ya que el tiempo de cálculo de la aplicación de filtros de convolución es menor. La entrada de una imagen integral en una localización $x = (x,y)$ está formada por la suma de los valores en la escala de grises de los pixeles de la imagen que se encuentran en un rectángulo formado por el origen y x.

Una vez que una imagen integral ha sido calculada, son necesarias tres operaciones más y cuatro accesos de memoria para calcular la suma de intensidades de un rectángulo de cualquier tamaño.

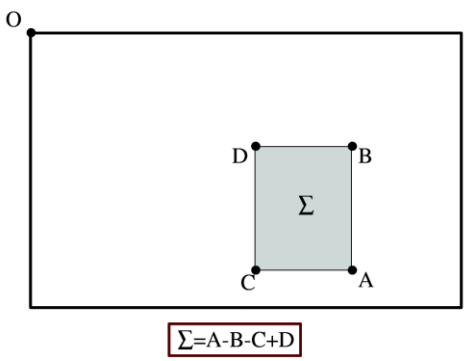


Figura 23: Cálculo de suma de intensidades para un rectángulo cualquiera. [3]

SURF basa su detector en matrices Hessianas debido a su gran precisión y bajo coste computacional. Pero a diferencia de otros detectores como el de Mikolajczyk y Schmid, SURF se basa en el determinante de la matriz para la localización de los puntos y la selección de la escala como hizo Lindeberg.

Así pues, dado un punto $x = (x, y)$ en una imagen I, la matriz Hessiana $\mathcal{H}(x, \sigma)$ en x a escala σ se define:

$$\boldsymbol{\mathcal{H}(x, \sigma)} = \begin{bmatrix} \boldsymbol{L_{xx}(x, \sigma)} & \boldsymbol{L_{xy}(x, \sigma)} \\ \boldsymbol{L_{xy}(x, \sigma)} & \boldsymbol{L_{yy}(x, \sigma)} \end{bmatrix}$$

Donde $\frac{\partial^2}{\partial x^2} g(\sigma)$ es la convolución de la derivada de segundo orden de la Gaussiana $L_{xx}(x, \sigma)$ con la imagen I en el punto x. Para el cálculo del determinante se realizan aproximaciones a las derivadas de segundo orden de la Gaussiana, obteniendo con ello las aproximaciones: D_{xx}, D_{xy}, D_{yy}. Para calcular el determinante de la matriz Hessiana que nos indicara la escala del punto será necesario aplicar la siguiente fórmula:

$$\mathbf{det}(\boldsymbol{\mathcal{H}_a prox}) = \boldsymbol{D_{xx} D_{yy}} - (\mathbf{0.9}\boldsymbol{D_{xy}})^{\mathbf{2}}$$

Como se ha mencionado previamente SURF guarda gran similitud con SIFT. En primer paso para obtener el descriptor de un punto de interés con SURF es calcular su orientación. Para obtener un punto invariante a la orientación se calcula el "Haar-wavelet" en las direcciones x e y en una región circular de radio $6s$, donde s es la escala del punto de interés. Una vez se ha calculado para todos los vecinos, se calcula la orientación dominante sumando todos los resultados dentro de una ventana deslizante que cubre un ángulo de de $\pi/3$.

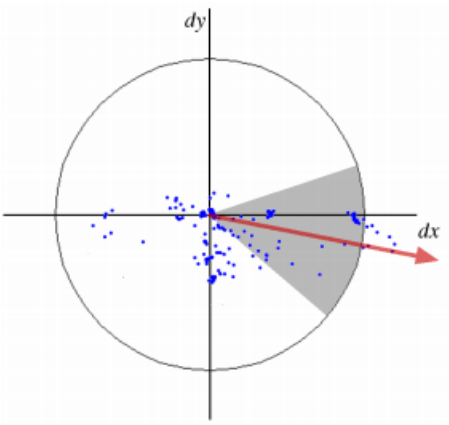


Figura 24: Ejemplo del cálculo de la orientación en SURF. [3]

Para construir el descriptor se utiliza una región cuadrada de tamaño $20s$ centrada en el punto de interés. Dicha región se divide en 4 subregiones, calculándose para cada una de ellas un conjunto de características simples. Posteriormente se calcula el "Haar-wavelet" para x e y y se suavizan los resultados mediante una Gaussiana, obteniéndose como resultado d_x y d_y. A su vez, para cada subregión se suman los resultados d_x y d_y además de calcularse su valor absoluto $|dx|$ y $|dy|$. De esta forma se dispone para cada subregión de un vector v compuesto por:

$$\boldsymbol{v} = (\sum \boldsymbol{d_x}, \sum \boldsymbol{d_y}, \sum |\boldsymbol{d_x}|, \sum |\boldsymbol{d_y}|)$$

El vector SURF se forma uniendo los diferentes vectores de las subregiones.

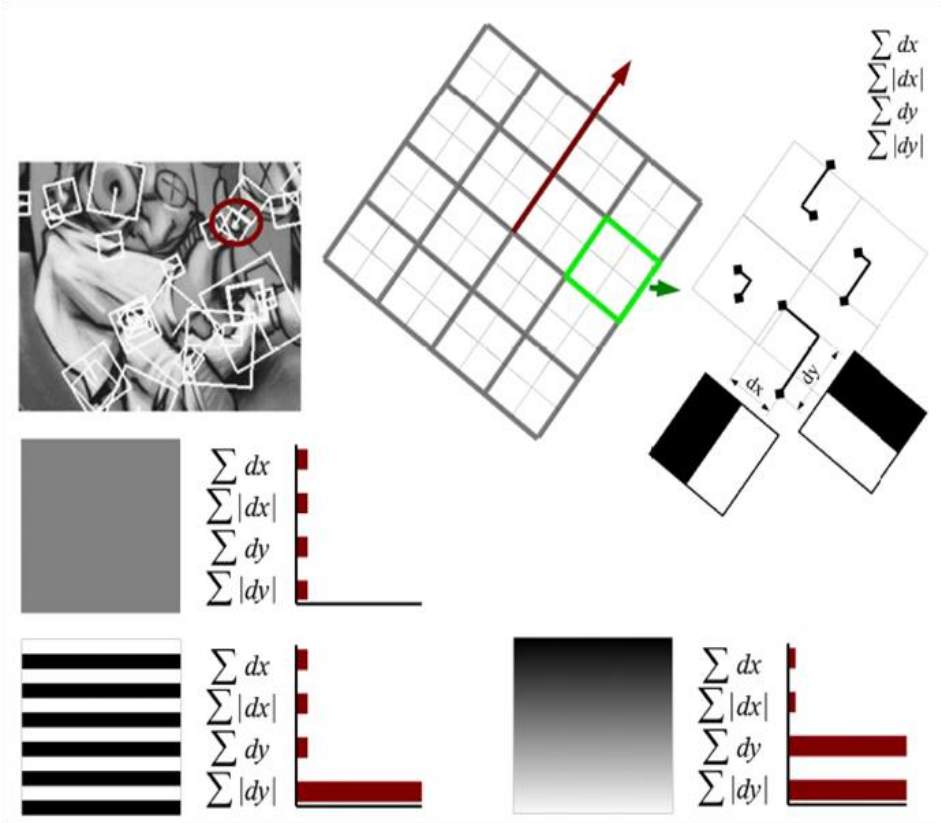


Figura 25: Ejemplo de los valores adquiridos por d_x y d_y para diferentes imágenes. [3]

4.3 MSER

MSER (Maximally Stable Extremal Regions) es una técnica para la detección de blobs en imágenes. Esta técnica fue propuesta por Matas [11] para encontrar correspondencias entre los elementos que aparecen en común en varias imágenes con distintos puntos de vista.

Funciona mediante thresholds, toma regiones de pixeles que contienen un nivel similar de grises, de forma que la diferencia entre los pixeles que forman las regiones se encuentra por debajo de un determinado threshold. Posteriormente se sustituyen estas regiones por elipses. Con las elipses obtenidas se procederá a utilizar otros algoritmos como SURF o SIFT para calcular los descriptores de dichas elipses y así poder comparar ambas imágenes.

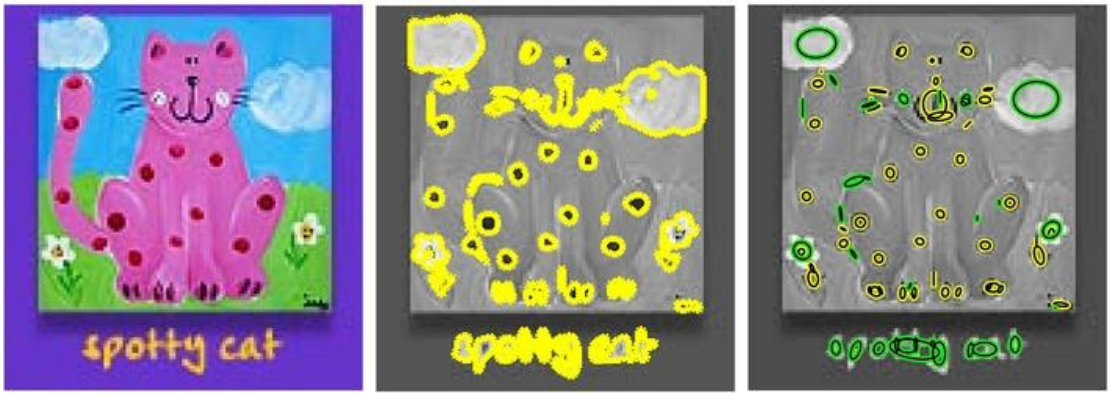


Figura 26: Ejemplo del resultado de aplicar MSER a una imagen. [14]

La idea básica del algoritmo que sigue MSER es la siguiente:

1. Se ordenan los pixeles de la imagen en orden de intensidad.
2. Los pixeles se añaden a un árbol en orden creciente de intensidad. Dicho árbol tiene las siguientes propiedades:
    - Todos los descendientes de un determinado pixel son un subconjunto que forma una región.
    - Todas las regiones son descendientes de algunos pixeles.
3. Las regiones son extraídas del árbol calculado dependiendo de los valores (tamaño mínimo de la región, tamaño máximo, etc.) y thresholds establecidos.
4. Las regiones duplicadas u otras regiones malas son eliminadas.
5. Se calculan las elipses correspondientes a las regiones obtenidas.

Debido a que con el algoritmo previamente expuesto se tiene una idea bastante general del funcionamiento de MSER se ha considerado que no era necesario profundizar en un mayor nivel de detalle.

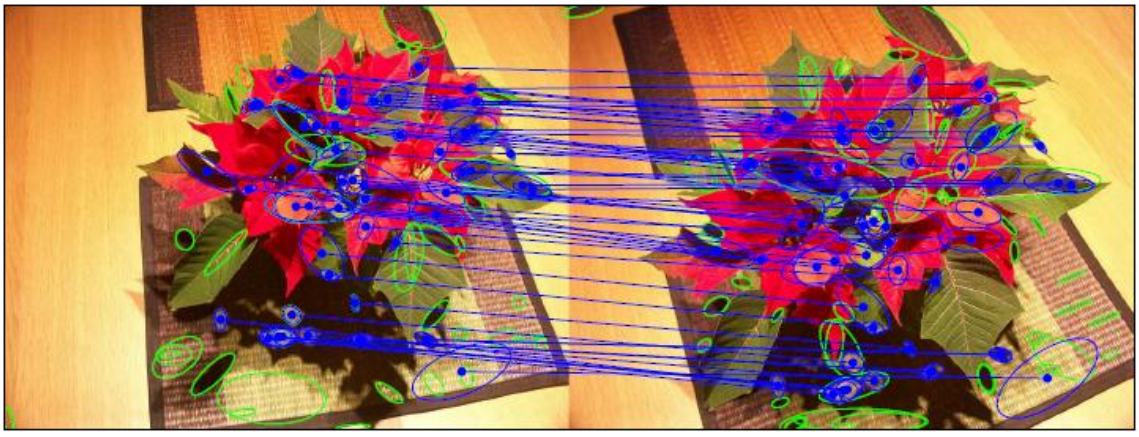


Figura 27: Ejemplo de emparejamiento entre dos imágenes con MSER. [14]

4.4 Comparativa

4.4.1 SIFT vs SURF

Tanto SIFT como SURF obtienen puntos de interés de la imagen invariantes a escala y orientación, así como a cambios de iluminación. Ambos obtienen además un vector descriptor por cada punto de interés, aunque cada uno lo calcula de una forma distinta.

Entre las diferencias encontramos que SIFT guarda la posición, escala y orientación debido a que en una misma posición (x, y) es posible encontrar varios puntos de interés con diferente escala s u orientación σ . En cambio en SURF para una posición (x, y) solo aparece un único punto de interés, por lo que no guarda la escala y orientación, aunque si la matriz de segundo momento y el signo de la laplaciana.

A continuación se muestra una comparativa entre el algoritmo SURF y SIFT utilizando en ambos descriptores de 128. Tanto SIFT como SURF trabajan con imágenes en formato pgm (escala de grises) por lo que ambos parten de la misma base.

Imagen	Puntos SIFT	Puntos SURF	Tiempo SIFT (ms)	Tiempo SURF (ms)
0000	1635	584	1879	494
0001	1943	659	2158	521
0002	2155	798	2483	572
0003	1967	673	2167	535
0004	1760	603	1932	501
0005	1548	529	1740	476
0006	1256	451	1656	466

Figura 28: Tabla comparativa entre SIFT y SURF.

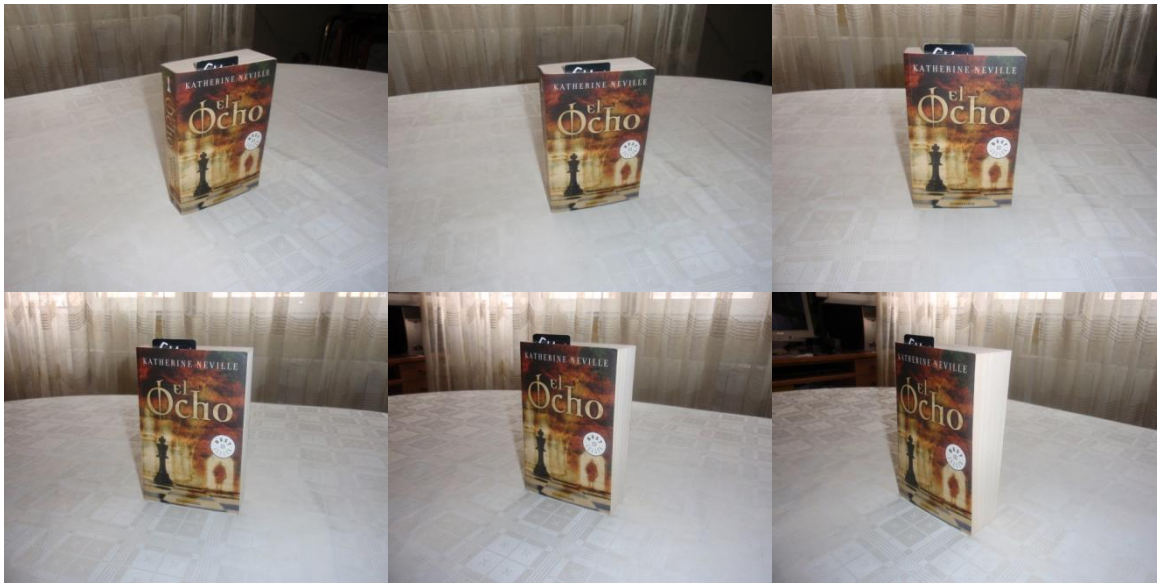


Figura 29: Imágenes utilizadas para la comparativa ordenadas de izquierda a derecha y de arriba abajo, siendo la imagen superior izquierda la 0000 y la inferior derecha la 0005.

En la tabla observamos como SURF es prácticamente 3 o 4 veces más rápido que SIFT. Mientras que en cuanto a puntos invariantes detectados SIFT obtiene más del doble de puntos que SURF. Sin embargo, tenemos que tener en cuenta que SIFT permite que existan varios puntos invariantes con distinta escala y orientación en una misma posición, mientras que en SURF solo sería posible uno. Es por ello que suponemos que el número de características detectada disminuiría si SIFT no permitiera varios puntos por posición. Así pues podemos deducir de la primera tabla que si nuestro objetivo es encontrar el mayor número de puntos de interés sin importarnos tanto el tiempo deberíamos optar por SIFT, mientras que si el tiempo es una prioridad para nosotros deberíamos elegir SURF.

Sin embargo, no podemos calificar SIFT y SURF únicamente por el número de puntos y el tiempo de ejecución. Sería necesario comprobar también la calidad de los puntos invariantes proporcionado por cada algoritmo, por lo que se realizaran pruebas de emparejamiento entre distintas imágenes para posteriormente comparar los resultados obtenidos entre SIFT y SURF.

Par de imágenes	Matches SIFT	Matches SURF	Matches SIFT tras RANSAC	Matches SURF tras RANSAC
0000-0002	172	178	78	61
0001-0002	378	256	287	189
0003-0002	421	294	356	217
0004-0002	351	224	278	187
0005-0002	94	97	42	36
0006-0002	46	56	24	14

Figura 30: Tabla 2 comparativa entre SIFT y SURF.

En la tabla 2 podemos visualizar como SIFT obtiene un numero de emparejamientos más elevado que SURF si las imágenes a emparejar están muy próximas, mientras que SURF encuentra más puntos si las imágenes comparadas se alejan de la inicial. Sin embargo, tras utilizar RANSAC en los emparejamientos obtenidos con SIFT y SURF observamos como el numero de emparejamientos correctos por parte de SURF es muy reducido comparado con el numero de emparejamientos correctos por parte de SIFT.

En resumen, según la comparativa realizada SURF tiene como principal ventaja su reducido coste computacional, mientras que SIFT obtiene un número mayor de puntos de interés además de un número menor de falsos positivos en los emparejamientos. Para verificar que las conclusiones obtenidas no dependían únicamente de las imágenes escogidas para la comparativa se decidió consultar el paper de A.M Romero [7] sobre comparativa de descriptores en donde se deducían unas conclusiones similares entre SIFT y SURF.

4.4.2 MSER

Otra posibilidad para realizar emparejamientos seria combinar SIFT o SURF con MSER. Para ello pasaríamos el algoritmo MSER a las imágenes que se quieren emparejar obteniendo con ello regiones o blobs con forma elíptica para cada imagen. Puesto que SIFT trabaja con regiones elípticas (SURF trabaja con regiones circulares por lo que es menos apto a combinar con MSER), el siguiente paso consistiría simplemente en calcular el descriptor SIFT en el centro de las elipses de las regiones proporcionadas por MSER con la escala y orientación correspondientes. Una vez obtenidos los descriptores solo quedaría compararlos entre sí, emparejando aquellos que se encuentren por debajo de un determinado umbral.

Al combinar MSER con SIFT se obtendría un número bastante reducido de puntos de interés comparado a si utilizáramos únicamente SIFT, reduciendo con ello el coste computacional. Sin embargo si nuestro objetivo es reducir el coste computacional podríamos utilizar únicamente SURF en lugar de MSER con SIFT, o incluso reducir aún más el coste computacional combinando MSER con SURF.

Entre los inconvenientes que existen por utilizar MSER encontramos que no funciona bien con imágenes con algún desenfoque de movimiento mientras que SIFT y SURF si funcionan correctamente.

4.4 Aplicación de emparejamiento

Para comprobar los resultados que obteníamos al utilizar SURF, SIFT y MSER a nuestras imágenes se procedió a desarrollar una aplicación de emparejamiento. En dicha aplicación básicamente se comparan los descriptores de una imagen con los de la otra, quedándonos con aquellos que más se asemejan y que cumplen ciertos criterios a nuestra elección. Una vez conocemos los emparejamientos se muestra por pantalla las dos imágenes, en donde a su vez aparecen líneas de diferentes colores (para facilitar la diferenciación entre líneas) que indican el punto de cada imagen que ha sido emparejado.

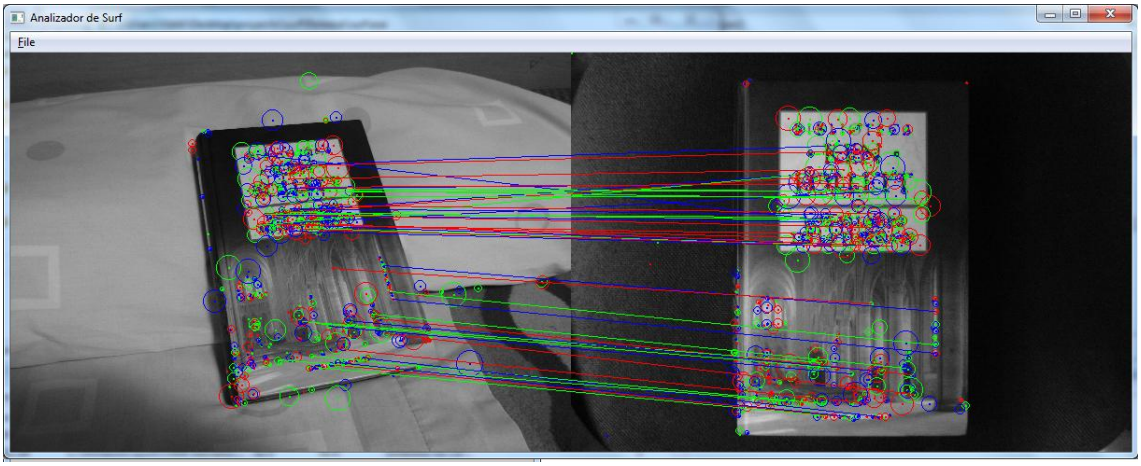


Figura 31: Ejemplo de la aplicación de emparejamiento.

4.4.1 Algoritmo desarrollado para el emparejamiento

Para cada descriptor i de la imagen 1 ***hacer***

buscar los dos descriptores de la imagen 2 que más se asemejan mediante la distancia euclidea (siendo j el descriptor j de la imagen 2 y k el descriptor k de la imagen 2).

si ((distEu(i,j) / distEu(i,k) < umbral1) & (distEu(i,j) < umbral2)) ***entonces***

Para el descriptor i' de la imagen 2, siendo i' == j ***hacer***

buscar los dos descriptores de la imagen 1 que más se asemejan mediante la distancia euclidea (siendo j' el descriptor j' de la imagen 1 y k' el descriptor k' de la imagen 1).

fin hacer

si ((distEu(j',i') / distEu(k',i') < umbral1) & (distEu(j',i') < umbral2) & (j' == i)) ***entonces***

emparejar descriptor(i) de la imagen 1 con el descriptor(j) de la imagen 2.

fin si

fin si

fin hacer

Para el cálculo de la distancia euclidea entre dos descriptores simplemente se utiliza la siguiente función, siendo p el descriptor de la imagen 1 y q el descriptor de la imagen 2.

$$\boldsymbol{d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \ldots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^{n} (p_{i1} - q_i)^2}.}$$

4.4.2 Explicación del algoritmo

Debido a que en este proyecto se busca como objetivo el posicionamiento de objetos en una escena y que para dicho objetivo se utiliza como principal información la lista de emparejamientos que nos proporciona este algoritmo, es recomendable reducir al máximo el número de posibles falsos positivos que pudiéramos obtener, aunque esto suponga reducir el número de emparejamientos. Así pues, preferimos sacrificar un mayor número de emparejamientos a cambio de que los que obtengamos sean de mayor calidad.

Para ello, inicialmente en el algoritmo buscamos para cada punto de interés de la imagen 1 aquellos dos puntos de interés de la imagen 2 cuyos descriptores tienen la menor distancia euclidea con respecto al descriptor del punto de interés de la imagen 1.

Una vez localizados comprobamos que al dividir la distancia euclidea del punto más próximo (j) por la distancia euclidea del segundo punto más próximo (k) hay al menos una diferencia inferior a un umbral1. Si se cumple esta condición significa que el punto más próximo tiene una distancia euclidea considerablemente más reducida que el segundo punto siendo así un buen candidato para el emparejamiento. Además se comprueba que la distancia euclidea del mejor punto es inferior a un determinado umbral2, garantizando así que los emparejamientos que se producen son de calidad.

Ya llegados a este punto se podría validar el emparejamiento, pero para ser algo más restrictivos y seguros preferimos realizar antes otra comprobación. En dicha comprobación hacemos exactamente lo mismo pero en sentido inverso, es decir, para el punto de interés candidato a emparejamiento de la imagen 2 buscamos los dos puntos de interés de la imagen 1 con menor distancia euclidea. Una vez localizados hacemos las mismas comprobaciones que antes, pero añadiendo una más en la cual se exige que el mejor punto de interés obtenido de la imagen 1 sea el mismo punto al cual estábamos buscando emparejamiento.

Con esta ultima comprobación se duplica el tiempo de cálculo a cambio de asegurarnos mejores emparejamientos y quitarnos algunos falsos positivos. En nuestro caso hemos preferimos realizar esta comprobación aunque aumente el tiempo de cálculo ya que necesitamos cerciorarnos de la calidad de los emparejamientos.

4.5 Resultados

4.5.1 Primeros Resultados

Para las siguientes imágenes se utilizo como descriptores SURF y SIFT, y como dataset el conjunto de imágenes realizadas en la cafetería del CPS.

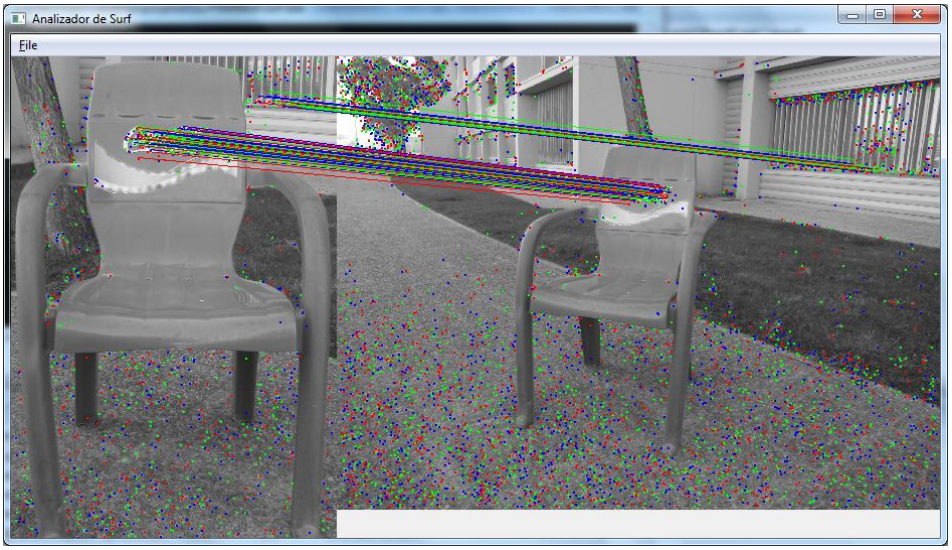


Figura 32: Primera prueba del algoritmo con SURF utilizando sillas en las imágenes.

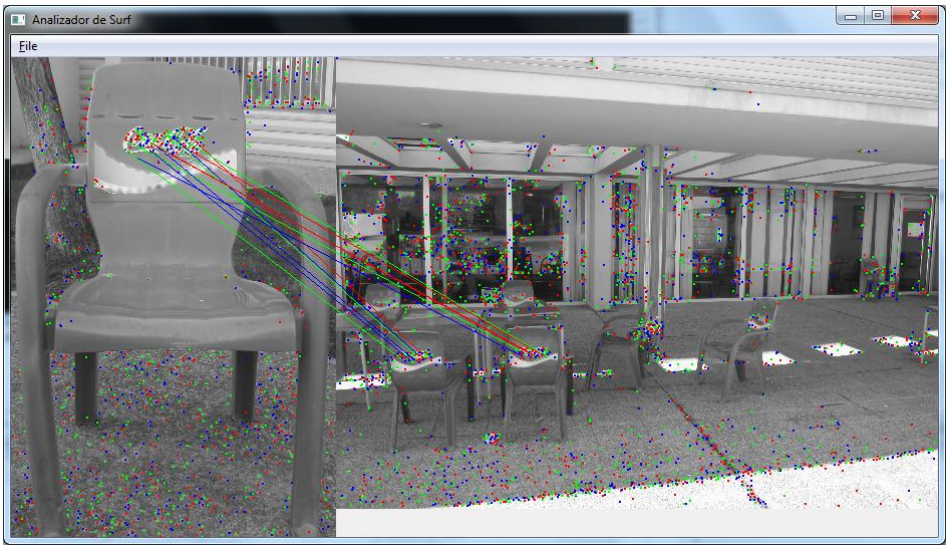


Figura 33: Segunda prueba del algoritmo con SURF utilizando sillas en las imágenes.

En las imágenes aparece mediante puntos de colores los puntos de interés que no han sido emparejados, mientras que los puntos emparejados aparecen unidos mediante líneas. En cuanto a los resultados obtenidos, podemos observar como SURF apenas obtiene puntos de interés en la silla, mientras que en los elementos que hay alrededor de ella aparecen bastantes puntos de interés. De hecho, los pocos puntos de interés que aparecen sobre la silla pertenecen al logotipo. Ya que la silla es prácticamente de un único color y apenas tiene detalles de los que se puedan obtener puntos característicos se decidió probar nuevos métodos para ver hasta qué punto utilizar esta silla como objeto iba a ser un impedimento para el proyecto.

La primera prueba consistía simplemente en utilizar SIFT en lugar de SURF, con este cambio se consiguió obtener más puntos de interés sobre la silla. Pero a la hora de realizar los emparejamientos con los descriptores obtenidos se producían muchos emparejamientos erróneos. La causa más probable de estos errores de emparejamientos vendrían por la gran similitud de los descriptores pertenecientes a la silla.

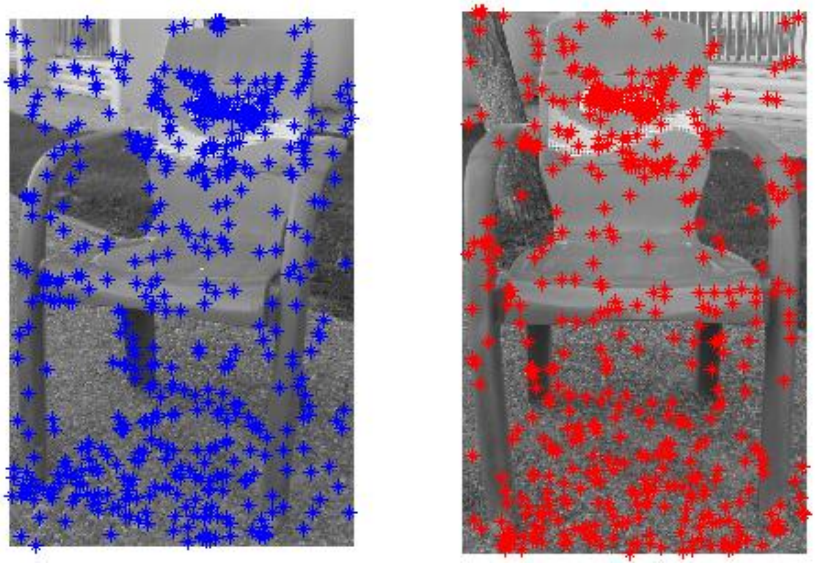


Figura 34: Ejemplo de puntos de interés obtenidos al aplicar SIFT.

Como segunda prueba se creó una malla de puntos de interés para dos imágenes, las cuales ya se había conseguido emparejar anteriormente con éxito mediante SURF y SIFT. El objetivo al utilizar esta malla era poder obtener puntos de interés de cada parte de la imagen, cosa que en las imágenes de la silla no se daba ya que los puntos se concentraban sobretodo alrededor de la silla, en lugar de sobre esta.

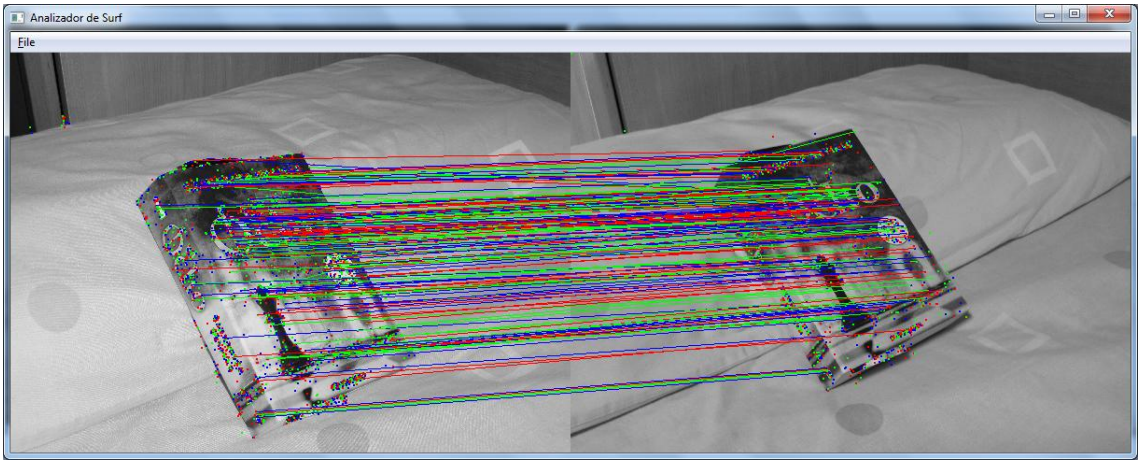


Figura 35: Ejemplo de emparejamiento con éxito mediante SURF.

Una vez se disponía de la posición de los distintos puntos de interés de la malla para cada imagen se procedió a calcular los descriptores SURF en dichas posiciones con unos determinados parámetros de radio y laplaciana. Los resultados obtenidos no fueron precisamente buenos por lo que se desecho esta opción rápidamente.

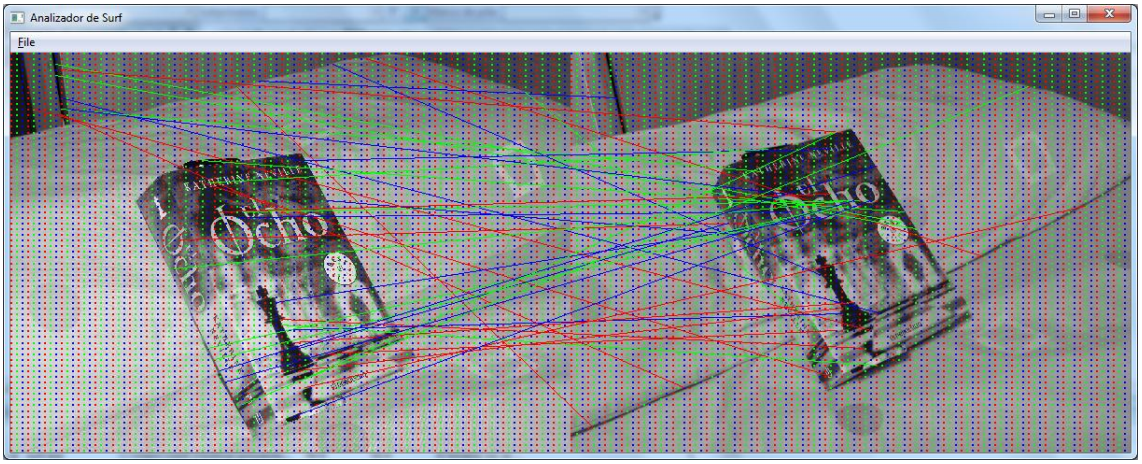


Figura 36: Segundo intento de emparejamiento mediante una malla de puntos de interés.

La tercera prueba consistía en hacer uso de los algoritmos por regiones o blobs como MSER para intentar obtener puntos de interés sobre la silla.

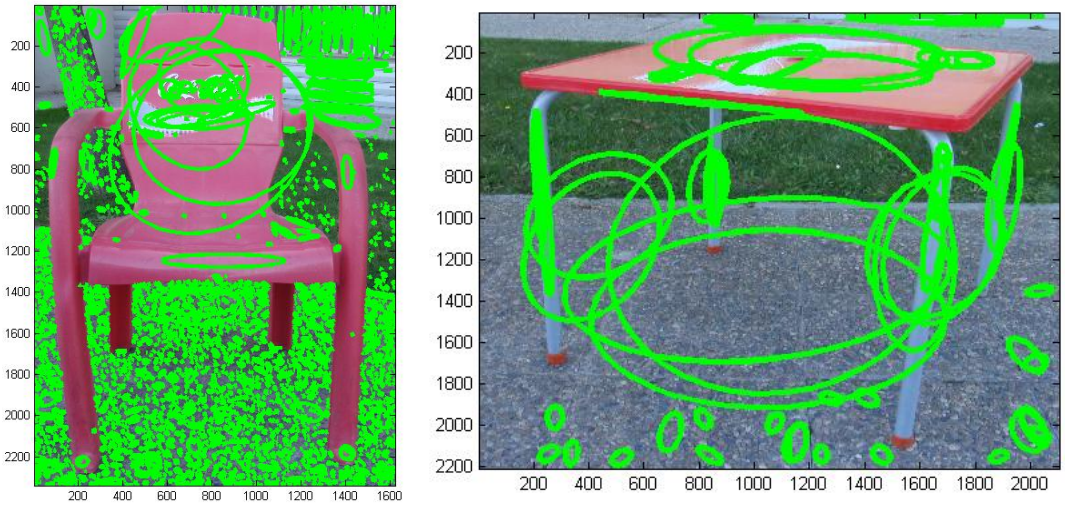


Figura 37: Ejemplo de aplicar MSER a una imagen de la silla y otra de la mesa.

Como se observa en la imagen MSER fue incapaz de obtener un gran número de regiones sobre la silla, mientras que alrededor de esta encontró numerosas regiones. En cuanto a la mesa, la situación es similar a la silla y MSER no encuentra suficientes regiones. Con estos resultados se decidió desechar también la idea de utilizar MSER.

Tras comprobar que con los algoritmos de cálculo de descriptores actuales no era posible obtener resultados satisfactorios para el primer dataset se procedió a utilizar el segundo dataset.

4.5.2 Resultados con el segundo dataset

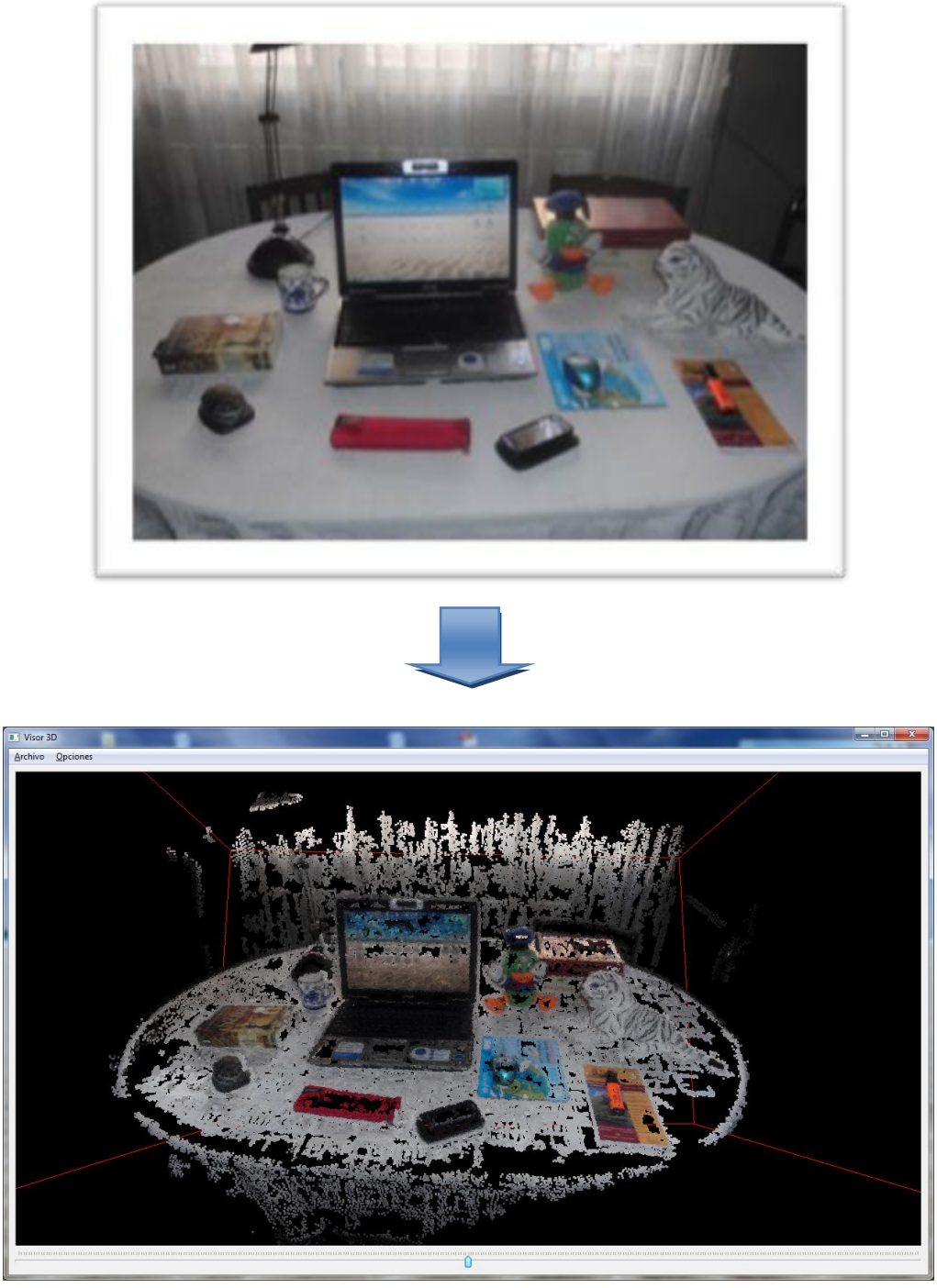


Figura 38: Escena reconstruida del segundo dataset.

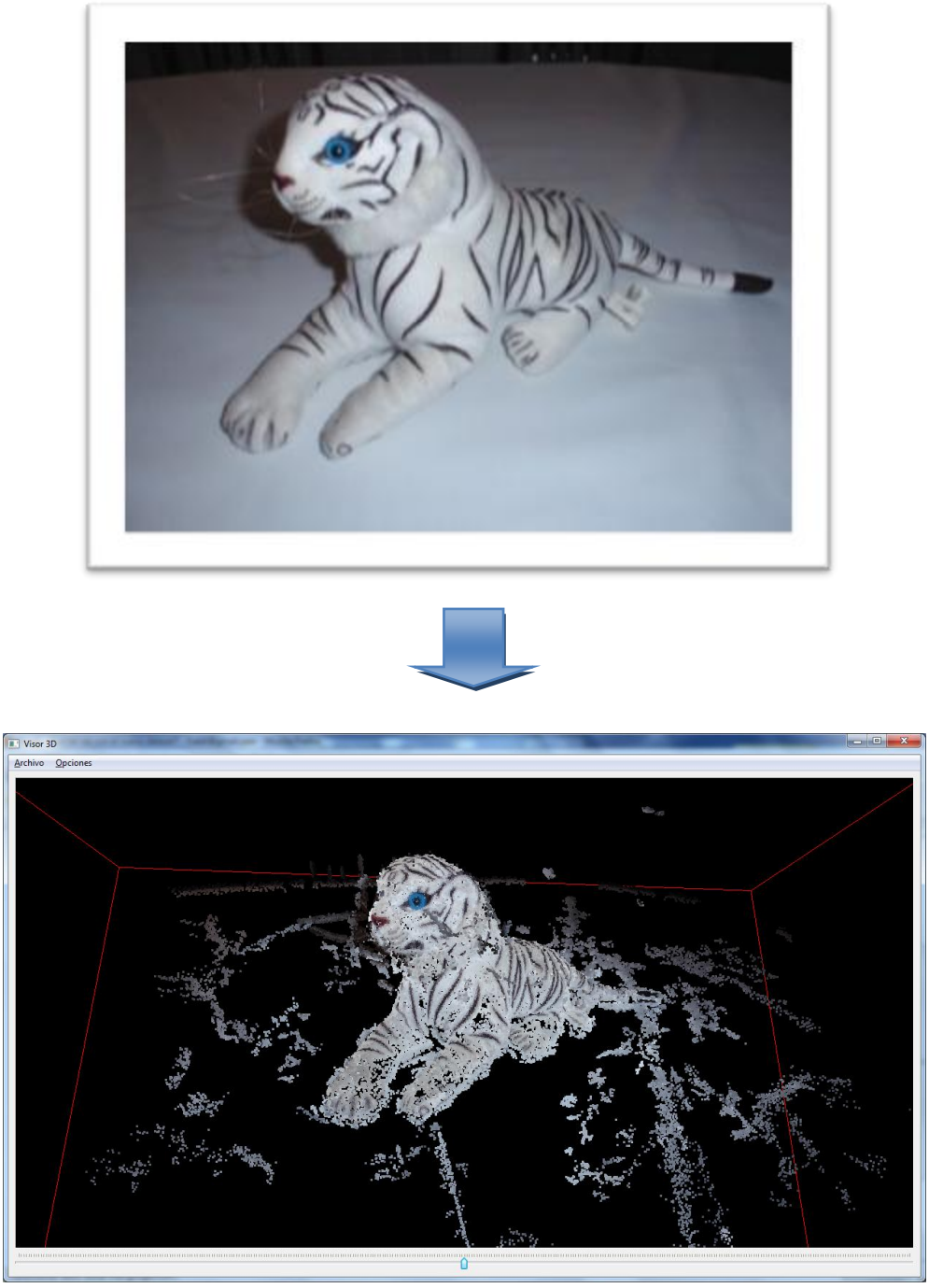


Figura 39: Objeto reconstruido del segundo dataset.

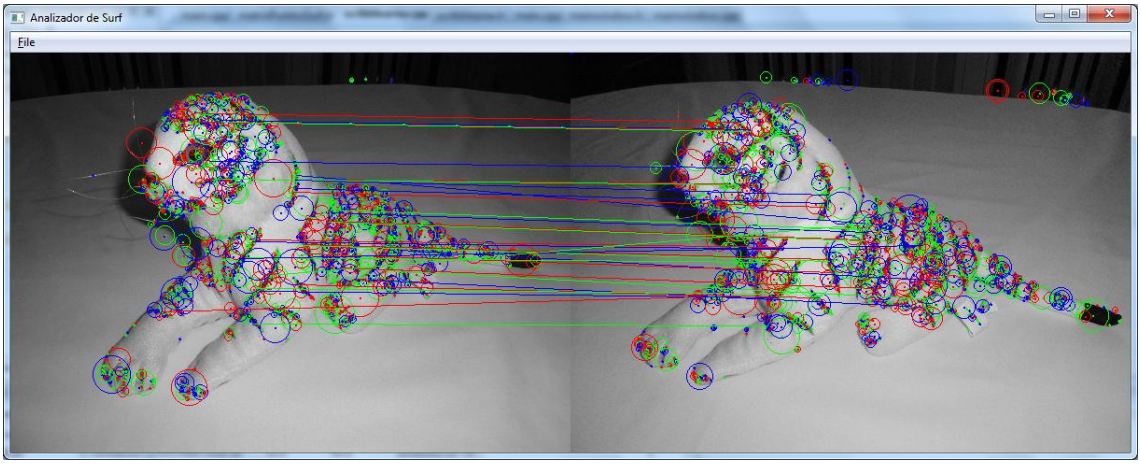


Figura 40: Ejemplo de emparejamiento con uno de los objetos del segundo dataset.

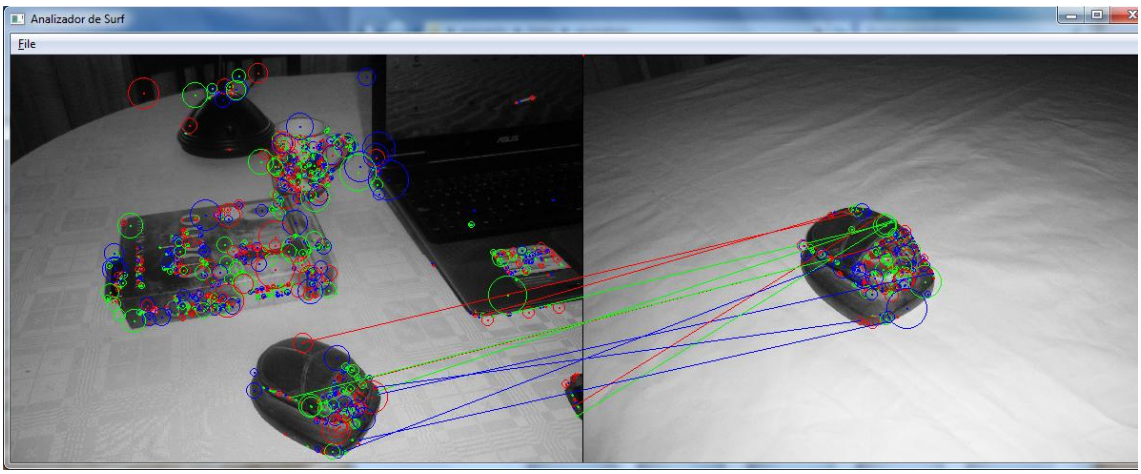


Figura 41: Ejemplo de emparejamiento con otro de los objetos del segundo dataset.

Tras realizar varias pruebas de emparejamiento tanto con SURF como con SIFT con los nuevos objetos de la escena comprobamos que se obtenían suficientes puntos de interés sobre los objetos a identificar además de obtener bastantes emparejamientos correctos. Por lo que se decidió continuar el proyecto con este nuevo dataset y dejar para más adelante o proyectos futuros el anterior dataset.

En las imágenes se percibe como existe algún falso positivo en los emparejamientos, sin embargo este problema no es excesivamente importante ya que en los procedimientos posteriores estos emparejamientos serán filtrados con el método RANSAC consiguiendo así eliminar los falsos positivos.

5. Posicionamiento de los objetos en la escena

Una vez disponemos ya de un dataset consistente con el que trabajar y de un algoritmo de emparejamiento con el que se obtienen resultados coherentes podemos ya proceder a programar los algoritmos necesarios para el posicionamiento de los objetos en la escena. Esta última fase del proyecto requiere de gran cantidad de cálculos, por lo que ha sido necesario dedicarle gran cantidad de tiempo tanto para programarla como para testearla y eliminar posibles errores. Algunas de las ecuaciones mostradas en esta sección del proyecto han sido ya vistas en la parte de reconstrucción 3D, esto es debido a que ambas fases guardan una gran similitud en sus cálculos.

Para posicionar el objeto en la escena con éxito es necesario seguir los siguientes pasos:

1. Calcular los emparejamientos entre una imagen de la escena y otra del objeto
2. A partir de los emparejamientos calcular la matriz fundamental
3. Calcular las matrices de calibración para cada cámara
4. Calcular la matriz esencial a partir de la matriz fundamental y las matrices de calibración
5. Descomponer la matriz esencial mediante SVD para obtener las cuatro posibles soluciones al combinar las dos matrices de rotación obtenidas y los dos vectores de translación
6. Proyectar o triangularizar un punto a 3D para cada una de las cuatro soluciones obtenidas y quedarnos con aquella solución que nos dé una profundidad Z positiva para ambas cámaras.
7. Calcular la escala entre la escena y el objeto
8. Calcular la escala de la translación obtenida
9. Posicionar el objeto en la escena con la solución correcta

Como ayuda a la hora de programar todos los pasos anteriores se ha decidido hacer uso de la librería OpenCV. Entre las ventajas que obtenemos al utilizar esta librería encontramos que contiene un potente algoritmo para el cálculo de la matriz fundamental, además de permitirnos calcular la descomposición SVD de una matriz y definir y trabajar matrices de forma relativamente sencilla.

A continuación se procede a explicar y describir con mayor detalle cómo se han desarrollado los diferentes puntos necesarios para el posicionamiento de objetos en la escena.

5.1 Calculo de la matriz fundamental

La matriz fundamental redefine la restricción epipolar, es decir la relación entre las dos proyecciones como :

$$\boldsymbol{x'^{T} * F * x = 0}$$

Siendo F la matriz fundamental, $x = [x\ y\ 1]^T$ el punto de la cámara izquierda y $x' = [x'\ y'\ 1]$ el punto de la cámara derecha emparejado con el punto x. Así pues para cada emparejamiento valido se deberá cumplir dicha propiedad.

Para calcular la matriz fundamental hacemos uso de la función cvFindFundamentalMat de la librería OpenCV. Dicha función necesita como parámetros de entrada las coordenadas en pixeles de los puntos emparejados de las imágenes que se quiere obtener la matriz fundamental. Además de los puntos hay que especificar el método de cálculo a utilizar, en nuestro caso optamos por el método de los 8 puntos con un filtrado RANSAC para la detección de espurios. En cuanto a los parámetros a utilizar para el filtrado RANSAC, hemos optado por un valor de 1.0 como la distancia máxima en pixeles de un punto a la línea epipolar y un valor de 0.99 como el nivel de seguridad deseado de que la matriz fundamental calculada sea correcta.

```
cvFindFundamentalMat( pointsObjeto, pointsEscena, F,
                                    CV_RANSAC, 1.0, 0.9999, status );
```

5.2 Calculo de la matriz esencial y las matrices de calibración

La matriz esencial puede ser calculada a partir de la matriz fundamental y las matrices de calibración de cada cámara mediante la fórmula:

$$\boldsymbol{E = K'^{T} * F * K}$$

Siendo F la matriz fundamental, K' la matriz de calibración de la cámara derecha y K la matriz de calibración de la cámara izquierda.

Antes de proceder a calcular la matriz esencial es necesario obtener las matrices de calibración de cada cámara. La matriz de calibración permite junto a la matriz de proyección de perspectiva transformar las coordenadas de la escena en tres dimensiones a pixeles en la imagen.

$$\widetilde{\boldsymbol{x_i}} = \boldsymbol{P} * \widetilde{\boldsymbol{X}}_{\boldsymbol{i}}$$

$$\widetilde{\boldsymbol{x_i}} = \boldsymbol{K} * \boldsymbol{P_{cam}} * \widetilde{\boldsymbol{X}}_{\boldsymbol{i}}$$

Siendo $\widetilde{x_i}$ las coordenadas del punto i en pixeles $(x\ y\ 1)$, K la matriz de calibración, P_{cam} la matriz de proyección en perspectiva y $\widetilde{X_i}$ las coordenadas del punto i en 3D $(x\ y\ z\ 1)$.

La matriz de proyección en perspectiva de la cámara izquierda la llamaremos $\boldsymbol{P_{cam}}$, mientras que la de la cámara derecha la llamaremos $\boldsymbol{P'_{cam}}$. Los valores de dichas matrices son los siguientes:

$$\boldsymbol{P_{cam}} = [\, \boldsymbol{I} \,|\, \boldsymbol{0} \,]$$

$$\boldsymbol{P'_{cam}} = [\,\boldsymbol{R} \mid \boldsymbol{t}\,]$$

Siendo I una matriz de identidad 3x3 y R y t la matriz de rotación y el vector de translación de la cámara derecha con respecto a la cámara izquierda.

La matriz de calibración tiene la siguiente forma:

$$\begin{bmatrix} \boldsymbol{\alpha_x} & \boldsymbol{s} & \boldsymbol{u_0} \\ \boldsymbol{0} & \boldsymbol{\alpha_y} & \boldsymbol{v_0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix}$$

Donde α_x y α_y son los factores de escala para cada eje de la imagen expresados en pixel/mm, s es el factor de torcimiento y u_0y $v_0$son el punto principal de la imagen en el nuevo sistema de coordenadas.

En nuestro caso se han asignado los siguientes valores a las matrices de calibración:

$$\boldsymbol{\alpha_x} = -\boldsymbol{f}$$
$$\boldsymbol{\alpha_y} = \boldsymbol{f}$$
$$\boldsymbol{s} = \boldsymbol{0}$$
$$\boldsymbol{u_0} = \boldsymbol{ancho\ de\ la\ imagen} / \boldsymbol{2}$$
$$\boldsymbol{v_0} = \boldsymbol{alto\ de\ la\ imagen} / \boldsymbol{2}$$

Donde f es la distancia focal de la cámara para esa foto, la cual obtenemos del archivo bundle.out generado por el Bundler. El factor de torcimiento es 0 ya que para el algoritmo de emparejamiento trabajamos con las imágenes que nos proporciona como salida Bundler, a las cuales ya se les ha aplicado un filtro para eliminar el factor de torcimiento de la cámara. u_0y $v_0$son la mitad del ancho y el alto de la imagen ya que el sistema de coordenadas pasa de estar en el centro de la imagen a estar en la esquina superior izquierda. Ha sido necesario a su vez negar α_x ya que el eje de coordenadas x cambia de dirección para los sistemas de referencia que empleamos.

Una vez disponemos de las matrices de calibración para la cámara izquierda y la cámara derecha podemos proceder a calcular la matriz esencial. Su cálculo es muy sencillo ya que solo hay que aplicar la fórmula previamente vista para obtenerla.

5.3 Descomposición SVD y cálculo de las cuatro soluciones

Para obtener las cuatro posibles matrices de proyección de perspectiva entre la cámara izquierda y derecha hay que realizar primero la descomposición SVD de la matriz esencial. Como se menciono previamente, OpenCV nos permite de forma sencilla obtener la descomposición SVD de una matriz mediante la función:

```
void cvSVD (CvArr* A, CvArr* S, CvArr* U=NULL, CvArr* V=NULL, int
            flags=0)
```

Donde A es la matriz que queremos descomponer y S, U y V son las matrices en las que se descompone. S debe ser una matriz diagonal con valores singulares, mientras que U y V son matrices ortogonales. La función pues descompone la matriz A en el producto de una matriz diagonal y dos matrices ortogonales que cumplen la siguiente fórmula:

$$\boldsymbol{A = U * S * V^T}$$

Tras descomponer la matriz esencial, ya podemos obtener la matriz de rotación y el vector de translación de una cámara respecto a otra. Las dos posibles matrices de rotación serian:

$$\boldsymbol{R_1 = U * W * V^T}$$

$$\boldsymbol{R_2 = U * W^T * V^T}$$

En cuanto al vector de translación, las dos posibles soluciones serian:

$$\boldsymbol{t_1 = u_3}$$

$$\boldsymbol{t_2 = -u_3}$$

Donde $u_3$corresponde a la última columna de la matriz U. La matriz W a utilizar para obtener la rotación tiene la siguiente forma:

$$\boldsymbol{W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}}$$

Con las matrices de rotación y los vectores de translación calculados obtenemos las siguientes posibles matrices de proyección de perspectiva entre la cámara izquierda y la cámara derecha:

$$\boldsymbol{P'_{cam} = [U * W * V^T \mid u_3]}$$

$$\boldsymbol{P'_{cam} = [U * W * V^T \mid -u_3]}$$

$$\boldsymbol{P'_{cam} = [U * W^T * V^T \mid u_3]}$$

$$\boldsymbol{P'_{cam} = [U * W^T * V^T \mid -u_3]}$$

El siguiente paso será comprobar cuál de estas cuatro matrices es la correcta.

5.4 Calculo de la solución correcta mediante proyección

Podemos determinar cuál de las cuatro soluciones obtenidas es la correcta mirando simplemente su interpretación geométrica.

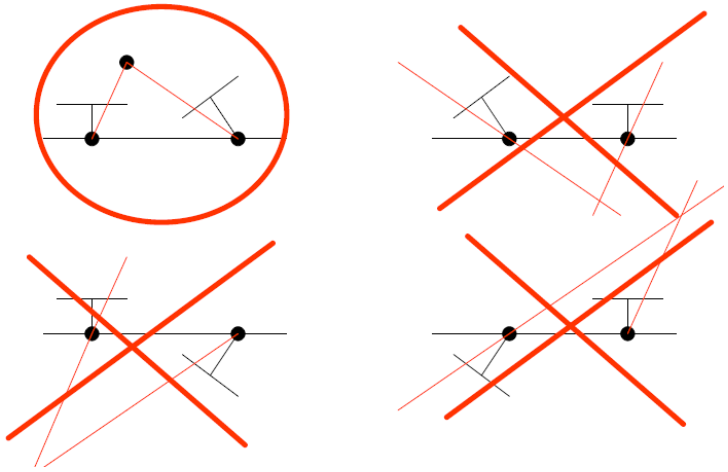


Figura 42: Ejemplo de solución correcta mediante interpretación geométrica.

El par correcto de cámaras será pues aquel que al proyectar un punto a tres dimensiones quede posicionado enfrente de las dos cámaras.

El procedimiento pues para obtener la solución correcta será el siguiente:

- Tomar un punto emparejado mediante el algoritmo de emparejamiento.
- Proyectar dicho punto a 3D.
- Calcular la profundidad del punto para cada cámara con cada solución.
- Elegir la solución que dé como solución un punto 3D con profundidad positiva para ambas cámaras.

Siendo $\widetilde{x_i}$ el punto en 2D para la cámara izquierda, P la matriz de proyección de la cámara izquierda, K la matriz de calibración de la cámara izquierda y P_{cam} la matriz de proyección de perspectiva de la cámara izquierda.

$$\widetilde{\boldsymbol{x}}_{\boldsymbol{i}} = \boldsymbol{P} * \widetilde{\boldsymbol{X}}_{\boldsymbol{i}}$$

$$\widetilde{\boldsymbol{x}}_{\boldsymbol{i}} = \boldsymbol{K} * \boldsymbol{P}_{\boldsymbol{cam}} * \widetilde{\boldsymbol{X}}_{\boldsymbol{i}}$$

$$\boldsymbol{K}^{-1} * \widetilde{\boldsymbol{x}}_{\boldsymbol{i}} = \boldsymbol{P}_{\boldsymbol{cam}} * \widetilde{\boldsymbol{X}}_{\boldsymbol{i}}$$

$$\boxed{\widetilde{\boldsymbol{x}}_{(\boldsymbol{i})\boldsymbol{cam}} = \boldsymbol{P}_{\boldsymbol{cam}} * \widetilde{\boldsymbol{X}}_{\boldsymbol{i}}}$$

Lo mismo para la cámara derecha:

$$\boxed{\widetilde{\boldsymbol{x}}'_{(\boldsymbol{i})\boldsymbol{cam}} = \boldsymbol{P}'_{\boldsymbol{cam}} * \widetilde{\boldsymbol{X}}'_{\boldsymbol{i}}}$$

A partir de estas formulas podemos obtener la proyección 3D de un punto, ya que conocemos $\tilde{x}_{(i)cam}$, $\tilde{x}'_{(i)cam}$, P_{cam} , P'_{cam} y lo que queremos obtener es $\tilde{X}_i$.

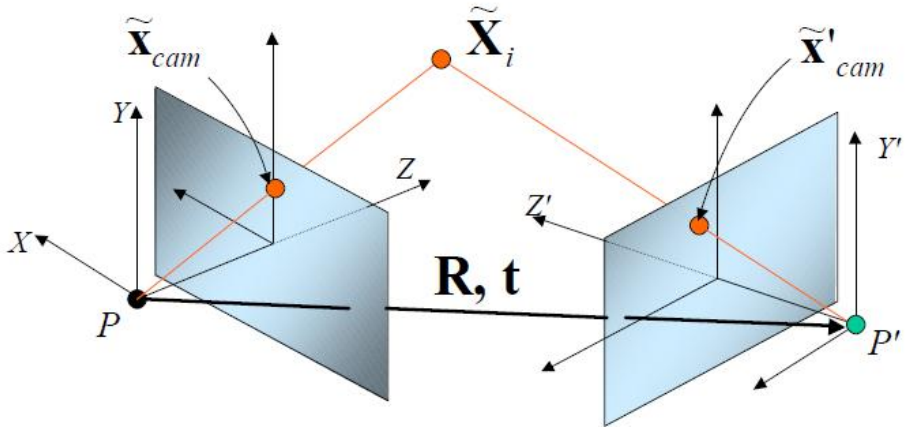


Figura 43: Ejemplo visual para el cálculo de la posición 3D de un punto.

La ecuación homogénea:

$$\widetilde{\boldsymbol{x}}_{(\boldsymbol{i})\boldsymbol{cam}} = \boldsymbol{P}_{\boldsymbol{cam}} * \widetilde{\boldsymbol{X}}_{\boldsymbol{i}}$$

Solo es correcta si se le añade un factor escala w.

$$\boldsymbol{w} * \begin{bmatrix} \boldsymbol{x_{icam}} \\ \boldsymbol{y_{icam}} \\ \boldsymbol{1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{P_{1cam}} \\ \boldsymbol{P_{2cam}} \\ \boldsymbol{P_{3cam}} \end{bmatrix} * \widetilde{\boldsymbol{X}}_{\boldsymbol{i}}$$

Donde P_{cami} son las filas de P_{cam} .

$$\boldsymbol{w} * \begin{bmatrix} \boldsymbol{x_{icam}} \\ \boldsymbol{y_{icam}} \\ \boldsymbol{1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{P_{1cam}} \\ \boldsymbol{P_{2cam}} \\ \boldsymbol{P_{3cam}} \end{bmatrix} * \widetilde{\boldsymbol{X}}_{\boldsymbol{i}} \Rightarrow \begin{cases} \boldsymbol{w} * \boldsymbol{x_{icam}} = \boldsymbol{P_{1cam}} * \widetilde{\boldsymbol{X}}_{\boldsymbol{i}} \\ \boldsymbol{w} * \boldsymbol{y_{icam}} = \boldsymbol{P_{2cam}} * \widetilde{\boldsymbol{X}}_{\boldsymbol{i}} \\ \boldsymbol{w} = \boldsymbol{P_{1cam}} * \widetilde{\boldsymbol{X}}_{\boldsymbol{i}} \end{cases}$$

Sustituyendo:

$$\Rightarrow \begin{cases} \boldsymbol{P_{3cam}} * \widetilde{\boldsymbol{X}}_{\boldsymbol{i}} * \boldsymbol{x_{icam}} = \boldsymbol{P_{1cam}} * \widetilde{\boldsymbol{X}}_{\boldsymbol{i}} \\ \boldsymbol{P_{3cam}} * \widetilde{\boldsymbol{X}}_{\boldsymbol{i}} * \boldsymbol{y_{icam}} = \boldsymbol{P_{2cam}} * \widetilde{\boldsymbol{X}}_{\boldsymbol{i}} \end{cases}$$

$$\Rightarrow \begin{cases} \boldsymbol{P_{3cam}} * \widetilde{\boldsymbol{X}}_{\boldsymbol{i}} * \boldsymbol{x_{icam}} - \boldsymbol{P_{1cam}} * \widetilde{\boldsymbol{X}}_{\boldsymbol{i}} = \boldsymbol{0} \\ \boldsymbol{P_{3cam}} * \widetilde{\boldsymbol{X}}_{\boldsymbol{i}} * \boldsymbol{y_{icam}} - \boldsymbol{P_{2cam}} * \widetilde{\boldsymbol{X}}_{\boldsymbol{i}} = \boldsymbol{0} \end{cases}$$

$$\Rightarrow \boxed{\begin{bmatrix} \boldsymbol{P_{3cam}} * \boldsymbol{x_{icam}} - \boldsymbol{P_{1cam}} \\ \boldsymbol{P_{3cam}} * \boldsymbol{y_{icam}} - \boldsymbol{P_{2cam}} \end{bmatrix} * \widetilde{\boldsymbol{X}}_{\boldsymbol{i}} = \boldsymbol{0}}$$

Similar para la otra cámara:

$$\Rightarrow \boxed{\begin{bmatrix} \boldsymbol{P'_{3cam}} * \boldsymbol{x'_{icam}} - \boldsymbol{P'_{1cam}} \\ \boldsymbol{P'_{3cam}} * \boldsymbol{y'_{icam}} - \boldsymbol{P'_{2cam}} \end{bmatrix} * \widetilde{\boldsymbol{X}}_{\boldsymbol{i}} = \boldsymbol{0}}$$

Combinando ambas soluciones:

$$\begin{bmatrix} \boldsymbol{P_{3cam} * x_{icam}} & \boldsymbol{- P_{1cam}} \\ \boldsymbol{P_{3cam} * y_{icam}} & \boldsymbol{- P_{2cam}} \\ \boldsymbol{P'_{3cam} * x'_{icam}} & \boldsymbol{- P'_{1cam}} \\ \boldsymbol{P'_{3cam} * y'_{icam}} & \boldsymbol{- P'_{2cam}} \end{bmatrix} * \widetilde{\boldsymbol{X}}_{\boldsymbol{i}} = \mathbf{0}$$

Esta solución tiene la forma $A * x = 0$,

donde $\boldsymbol{A} = \begin{bmatrix} \boldsymbol{P_{3cam} * x_{icam}} & \boldsymbol{- P_{1cam}} \\ \boldsymbol{P_{3cam} * y_{icam}} & \boldsymbol{- P_{2cam}} \\ \boldsymbol{P'_{3cam} * x'_{icam}} & \boldsymbol{- P'_{1cam}} \\ \boldsymbol{P'_{3cam} * y'_{icam}} & \boldsymbol{- P'_{2cam}} \end{bmatrix}$

Antes de proceder a solucionar el sistema anterior normalizamos A,

$$\boldsymbol{A_{norm}} = \begin{bmatrix} \boldsymbol{\frac{1}{\|A_1\|} A_1} \\ \boldsymbol{\frac{1}{\|A_2\|} A_2} \\ \boldsymbol{\frac{1}{\|A_3\|} A_3} \\ \boldsymbol{\frac{1}{\|A_4\|} A_4} \end{bmatrix}$$

Una vez tenemos A normalizado podemos obtener fácilmente de forma rápida $\tilde{X}_i$, ya que $\tilde{X}_i$ es equivalente a la última columna de V donde $U * S * V^T = A_{norm}$ es la descomposición SVD de A_{norm} .

Tras calcular el punto 3D $\tilde{X}_i$, podemos comprobar cuál de las cuatro posibles matrices de proyección previas es la correcta. Para ello simplemente basta con observar el resultado de las siguientes ecuaciones para las cuatro soluciones:

$$\widetilde{\boldsymbol{X}}'_{\boldsymbol{i}} = \boldsymbol{P} * \widetilde{\boldsymbol{X}}_{\boldsymbol{i}}$$

$$\widetilde{\boldsymbol{X}}''_{\boldsymbol{i}} = \boldsymbol{P}' * \widetilde{\boldsymbol{X}}_{\boldsymbol{i}}$$

Si la componente Z de $\tilde{X}'_i$ y $\tilde{X}''_i$ es positiva significa que el punto 3D calculado está enfrente de las dos cámaras y por lo tanto es la solución correcta.

5.5 Calculo de la escala entre escena y objeto

Para el cálculo de la escala entre la escena y el objeto vamos a utilizar algunos datos que nos proporciona el Bundler. En nuestro caso, los puntos de interés que utilizamos para trabajar con cada imagen los obtenemos de los que calculo previamente el Bundler para conocer la posición de las cámaras. Además de proporcionarnos los puntos de interés y las posiciones de las cámaras, Bundler guarda en un fichero la posición 3D de algunos de los puntos que ha sido capaz de reconstruir antes de la fase de PMVS. Esta información es muy valiosa ya que es la que nos va a permitir conocer la escala entre la escena y el objeto.

Nuestro objetivo es utilizar el listado de emparejamientos que hemos calculado anteriormente para consultar si en la lista de puntos 3D del Bundler aparece la posición 3D de ese punto. Para conocer la escala de la escena o el objeto basta con saber la posición 3D de dos emparejamientos y calcular la distancia entre estos.

Así pues los pasos a seguir serian los siguientes:

- Recorrer la lista de emparejamientos obtenida entre una imagen de la escena y otra del objeto y consultar para cada imagen si en el fichero

que nos proporciona Bundler se conoce la posición 3D del par de puntos emparejados.

- Buscar otro par de puntos distinto al anterior en la lista de emparejamientos de los que se conozca su posición 3D.
- Siendo A y A' el par de puntos emparejados en el primer paso y B y B' el par de puntos emparejados en el segundo paso, donde A y B corresponden a puntos de interés de la imagen de la escena y A' y B' corresponden a puntos de interés de la imagen del objeto. Calcular la distancia de los puntos en 3D entre A y B para conocer la escala de la escena. Calcular la distancia entre los puntos en 3D entre A' y B' para conocer la escala del objeto.

Una vez conocida la escala de la escena y del objeto podemos posicionar el objeto con el tamaño adecuado a la escena simplemente multiplicando los puntos 3D del objeto a posicionar por $escala_{esc}\,/escala_{obj}$.

5.6 Calculo de la escala de la translación

Para obtener la escala de la translación se debe calcular la proyección 3D de los puntos obtenidos para el cálculo de la escala de la escena y el objeto. Para proyectar los puntos a 3D vamos a realizar los mismos pasos que hemos realizado para calcular cual de las cuatro soluciones era la correcta, con la diferencia de que en lugar de utilizar P_{cam} , P'_{cam} , $\tilde{x}_{icam}$, $\tilde{x}'_{icam}$ vamos a utilizar P , P', $\tilde{x}_i$, $\tilde{x}'_i$.

Es decir A será:

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{P_3} * \boldsymbol{x_i} - \boldsymbol{P_1} \\ \boldsymbol{P_3} * \boldsymbol{y_i} - \boldsymbol{P_2} \\ \boldsymbol{P'_3} * \boldsymbol{x'_i} - \boldsymbol{P'_1} \\ \boldsymbol{P'_3} * \boldsymbol{y'_i} - \boldsymbol{P'_2} \end{bmatrix}$$

Una vez tengamos A normalizada, haremos la descomposición SVD:

$$U * S * V^T = A_{norm}$$

De donde la última columna de V corresponderá al punto 3D proyectado. Cuando dispongamos ya de la posición 3D de ambos emparejamientos procederemos a calcular la distancia entre estos. Para saber la escala de la translación bastara con dividir la escala de la escena por la distancia que acabamos de obtener.

Con la escala de la translación obtenida bastara con multiplicar la translación de una cámara con respecto a otra por dicha escala y así posicionar correctamente el objeto.

5.7 Posicionamiento del Objeto

Una vez realizados todos los cálculos anteriores deberíamos ser capaces de posicionar un objeto en la escena sin problema. Para posicionar el objeto disponemos de los siguientes datos:

- Modelo de la escena y del objeto (Los coordenadas de los puntos 3D de cada modelo están referenciados cada uno con respecto a una referencia mundo).
- Posición de la cámara de la escena respecto a la referencia mundo del modelo (matriz de rotación R_{wEsc}^{camEsc} y vector de translación t_{wEsc}^{camEsc} proporcionado por Bundler en la reconstrucción 3D no densa).
- Posición de la cámara del objeto respecto a la referencia mundo del modelo (matriz de rotación R_{wObj}^{camObj} y vector de translación t_{wObj}^{camObj} proporcionado por Bundler en la reconstrucción 3D no densa).
- Posición de la cámara de la escena respecto a la del objeto (matriz de rotación R_{camObj}^{camEsc} y vector de translación t_{camObj}^{camEsc} calculados mediante SVD partiendo de la matriz fundamental obtenida a partir de los emparejamientos entre la imagen de la escena y la imagen del objeto).
- Escala del modelo del objeto ($escala_{obj}$) y del modelo de la escena $escala_{esc}$.
- Escala del vector de translación de la cámara de la escena respecto a la del objeto ($escala_t$).

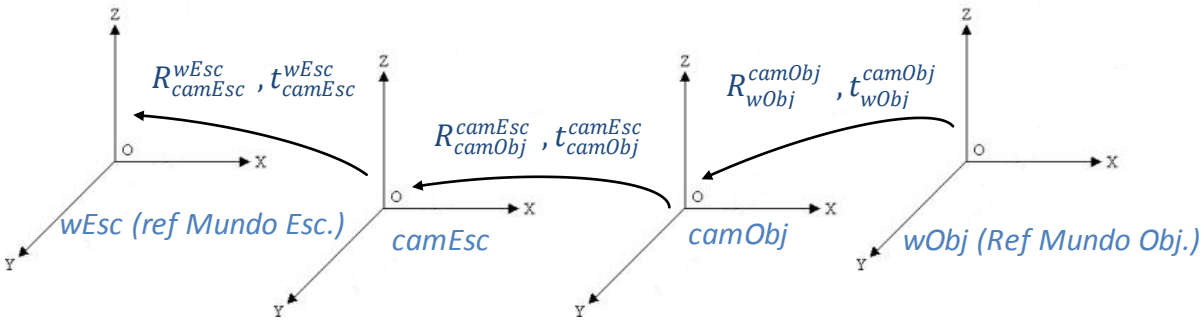


Figura 44: Ejemplo de transformaciones seguidas durante el posicionamiento.

Para posicionar correctamente el objeto cogeremos cada punto $\vec{X}_i = (x, y, z)$ del modelo 3D del objeto y:

1. Lo multiplicamos por $escala_{esc} / escala_{obj}$ para tener el objeto a la misma escala que aparece en la escena.
2. Después lo multiplicamos por R^{camObj}_{wObj} y le sumaremos la translación t^{camObj}_{wObj}. Con este paso hemos pasado de tener $\vec{X}_i$ referenciado con respecto al mundo del modelo del objeto a tenerlo referenciado con respecto a la cámara del objeto.
3. Multiplicamos $\vec{X}_{icamObj}$ por R^{camEsc}_{camObj} y le sumamos $t^{camEsc}_{camObj} * escala_t$. Ahora tenemos las coordenadas del punto referenciadas con respecto a la cámara escena.
4. Solo queda pasar $\vec{X}_{icamEsc}$ a las coordenadas mundo del modelo de la escena. Para ello le restamos t^{camEsc}_{wEsc} a $\vec{X}_{icamEsc}$ y lo multiplicamos por $R^{camEsc\ -1}_{wEsc}$. Con este último paso ya tenemos el punto en las coordenadas de la escena y por lo tanto solo falta pintarlo por pantalla para comprobar los resultados.

5.8 Resultados

A continuación se muestra un ejemplo de los resultados obtenidos tras posicionar un objeto en la escena. En este ejemplo se utilizan la escena y los objetos del segundo dataset. El objeto a posicionar en la escena va a ser el tigre parcialmente incompleto y que se localiza en la parte derecha de la mesa (figura 45).

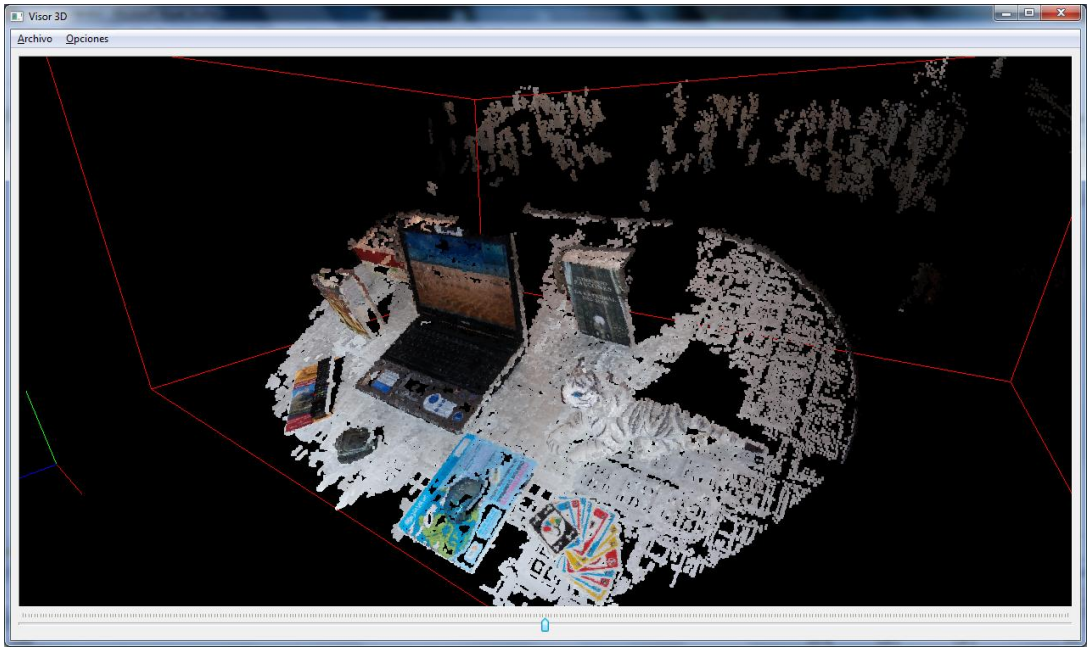


Figura 45: Imagen de la escena a utilizar.

En la figura 46 observamos el modelo del tigre a posicionar en la mesa. A diferencia del modelo del tigre que se visualiza en la mesa, de este modelo se tiene su completa reconstrucción a 360º, por lo que tras posicionar este modelo en la escena se podra visualizar partes del tigre que antes no se visualizaban.

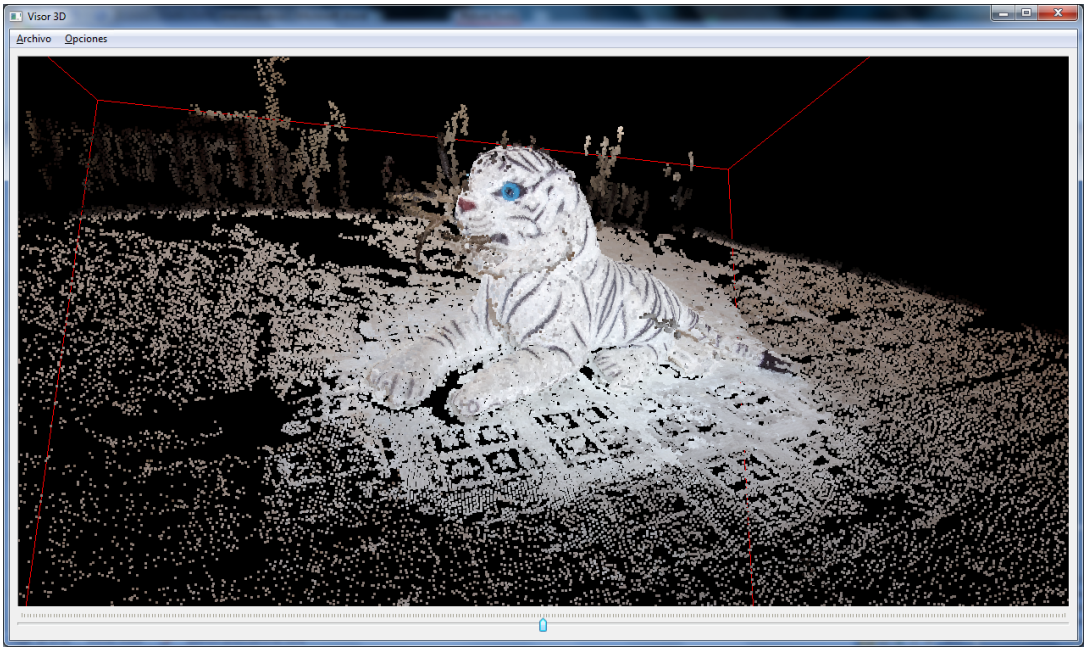


Figura 46: Imagen del objeto a utilizar.

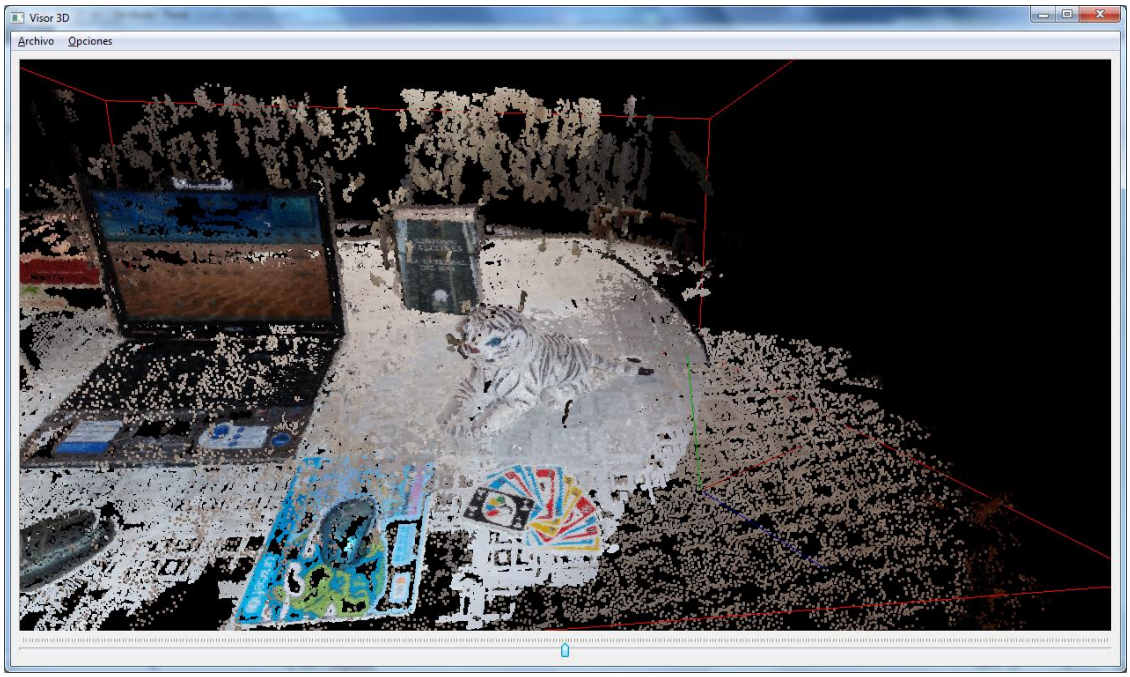


Figura 47: Resultado de posicionar el modelo del objeto sobre la escena.

Figura 48: Resultado de posicionar el modelo del objeto sobre la escena.

En la figura 48 se observa que el posicionamiento del objeto no es exacto, para ello basta que nos fijemos en la zona de la cola del tigre para visualizar más claramente el error. Este error es normal ya que la matriz fundamental calculada no es exacta y a partir de esta se han ido realizado numerosos cálculos que han podido incrementar o modificar dicho error. Sin embargo, se podría una vez realizada esta primera aproximación realizar algún calculo posterior para terminar de posicionar el objeto en su posición correcta.

6. Trabajo Futuro

Como trabajo futuro para este proyecto se podrían realizar numerosas mejoras entre las que encontramos las siguientes:

6.1 Optimización mediante kd-tree del algoritmo de emparejamiento

En este proyecto se ha programado un algoritmo de emparejamiento que devuelve resultados satisfactorios a costa de un tiempo medianamente elevado. Sin embargo este tiempo de cálculo podría ser reducido considerablemente si para el cálculo de los emparejamientos se utilizaran los arboles kd-tree. El motivo por el cual no se emplearon inicialmente en este proyecto fue debido a la falta de tiempo para su integración e implementación y a que al fin y al cabo el objetivo del proyecto no se veía afectado por la utilización o no de estos.

La idea básica de la utilización de kd-tree es poder introducir los diferentes descriptores de la imagen de la escena y del objeto cada uno en su correspondiente árbol kd-tree. Una vez disponemos de un árbol con los descriptores de la escena y de otro con los descriptores del objeto podemos proceder a buscar los descriptores que más se asemejan a un descriptor en un tiempo mucho más reducido.

6.2 Correspondencia con bolsa de palabras

En nuestro caso hemos trabajado con un pequeño dataset de objetos y por lo tanto no hemos tenido muchos inconvenientes para trabajar con él. Sin embargo en un futuro dicho dataset podría ser de un tamaño considerable y llegar a convertirse en un quebradero de cabeza, debido al tiempo que requiere comparar las diferentes imágenes de la escena con las de los objetos que tenemos registrados en nuestra base de datos para saber que objetos aparecen en dicha escena.

Una de las formas de conseguir reducir estos tiempos de cálculo seria la utilización de las denominadas bolsas de palabras. La idea básica consiste en que muchos de los descriptores de un objeto guardan una gran similitud, por lo que se podría asignar a cada objeto de la base de datos unos descriptores básicos que serian aquellos que mejor describan el objeto y por lo tanto más predominan en este. De esta forma bastaría con comparar las imágenes de la escena con los descriptores básicos de cada objeto en lugar de con todas las imágenes de cada objeto para saber si existe la posibilidad de que aparezca ese objeto en la escena.

6.3 Mejorar el posicionamiento

Como se ha mencionado previamente, el posicionamiento del objeto en la escena no es exacto y depende en gran medida de la calidad de los emparejamientos calculados antes de posicionar el objeto además de otros factores como podrían ser posibles fallos en la distancia focal, factores de distorsión etc. Sin embargo los resultados obtenidos suelen ser muy próximos a la posición exacta del objeto consiguiendo así una buena primera aproximación.

Como trabajo futuro se podría intentar realizar una segunda aproximación a partir de esta primera para intentar mejorar los resultados. Una posible forma de realizar una segunda aproximación seria coger varios puntos 3D de la escena y del objeto que hayan sido previamente emparejados y desplazar el objeto para reducir la distancia de los puntos emparejados a cero.

6.4 Escenas con objetos duplicados

En este proyecto se utilizo como escena la cafetería del CPS donde aparecían numerosas sillas y mesas esparcidas a lo largo de esta. Sin embargo se desecho debido a que las sillas y mesas no daban suficiente información como para su posicionamiento en la escena. La escena tenía como característica principal que en ella aparecían objetos duplicados y por lo tanto era necesario aplicar una fase previa a la de

emparejamiento para evitar que por ejemplo al intentar emparejar una foto de la cafetería con otra foto de una silla se emparejaran los puntos de interés de la foto de la silla con por ejemplo 9 sillas de la foto de la escena.

Una forma de conseguir trabajar con imágenes en las que aparecen objetos duplicados podría ser dividir la escena en pequeñas regiones para intentar que en cada región aparezca únicamente una vez el objeto, sin embargo si el objeto duplicado está superpuesto o muy próximo a otro igual habría que pensar otro posible método. También se podría intentar aprovechar el conocimiento de las distancias y posiciones de los puntos de interés del objeto para intentar saber si en una región aparece algún objeto duplicado.

6.5 Escenas con objetos menos flexibles a puntos de interés

Otra posible mejora a este proyecto consistiría en hacerlo más robusto frente a datasets mas complejos. En los resultados de este proyecto se han utilizado escenas y objetos medianamente sencillos con los cuales se esperaba que SIFT obtuviera sin problema un considerable número de puntos de interés que permitieran obtener sin dificultad una matriz fundamental con errores reducidos para conseguir un posicionamiento aceptable.

En nuestro caso se ha decidido utilizar como método para reconocer objetos en una escena la comparación de puntos de interés entre imágenes, pero podría haber sido perfectamente válido utilizar los puntos 3D del modelo. Cuando se decidió trabajar con el primer dataset se observo que las sillas no disponían de suficientes puntos de interés pero sin embargo gracias a los objetos del entorno a la silla fue posible obtener la posición de las cámaras y con ello el modelo 3D de la silla. Con unos algoritmos correctos se podría utilizar la información de los modelos 3D de los objetos de la base de datos para compararlos con el modelo de la escena e intentar saber si aparecen o no dichos objetos y así proceder a posicionarlos.

6.6 Eliminación de los puntos anteriores del objeto en la escena

En este proyecto nos hemos limitado a insertar el objeto en el modelo de la escena sin borrar los puntos que había de dicho objeto previamente en la escena. Sin embargo se podría mejorar los resultados finales si se borraran los puntos próximos a la localización final del objeto mediante algún algoritmo o se buscaran y eliminaran puntos similares al objeto a insertar en la escena.

Una cuestión a tener en cuenta en caso de desarrollarse esta mejora seria que en los modelos empleados por nosotros para los objetos aparecen además de los objetos parte del escenario en el que se realizaron las fotos. Sería posible desarrollar una aplicación en la que se pudiera desplazar un cubo o malla que indicara los límites del modelo de forma que los puntos que se encontraran fuera serian excluidos mientras que el resto serian los que realmente formaran parte del modelo.

7. Conclusiones

7.1 Resumen y Conclusiones básicas

A lo largo de este proyecto se ha buscado como objetivo comparar y reconocer los objetos que aparecen en una escena 3D con los objetos que tenemos catalogados en una pequeña base de datos. Una vez reconocido un objeto en la escena el segundo objetivo del proyecto ha sido posicionar el modelo del objeto de la base de datos en el mismo lugar que aparece en la escena. Gracias al primer objetivo podemos obtener cierta información de la escena y catalogarla semánticamente dependiendo de los objetos que en esta aparezcan, mientras que con el segundo objetivo podemos mejorar el aspecto de la escena 3D sustituyendo objetos incompletos en esta por modelos completos de la base de datos consiguiendo así visualizar partes de la escena de las que no se tenía información al obtener el modelo 3D.

Tras la realización de este proyecto se ha conseguido llevar a cabo con éxito ambos objetivos utilizando para ello una escena de características sencillas en la cual no aparecían objetos duplicados y en la cual los objetos proporcionaban suficientes información como para su reconocimiento y posicionamiento.

En las pruebas realizadas a lo largo de este proyecto se ha comprobado que al posicionar objetos comparando imágenes que tenían un pequeño grado de variación en el ángulo de visión los objetos se posicionaban con un grado de error mínimo, mientras que al comparar imágenes con un mayor variación en su ángulo de visión el grado de error en el posicionamiento se incrementaba. Este suceso en principio es independiente de la implementación realizada y se considera que tiene su explicación lógica, ya que cuanto más difiere una imagen de otra más difícil es encontrar emparejamientos correctos y por lo tanto la matriz fundamental calculada es menos

precisa produciéndose así mayores errores entre la matriz de rotación y el vector de translación reales y los calculados por la descomposición SVD de la matriz esencial.

En este proyecto se han encontrado también numerosos problemas para posicionar objetos que proporcionaban pocos puntos de interés al aplicarles SIFT u objetos con descriptores de extrema similitud, viéndonos obligados a prescindir del primer dataset y utilizar únicamente el segundo. Como se ha mencionado previamente, dependemos considerablemente del numero de emparejamientos correctos que se le proporcionan al algoritmo de cálculo de la matriz fundamental ya que si estos son escasos o están todos agrupados en una zona de la imagen la matriz fundamental no será precisa y por lo tanto el posicionamiento del objeto en la escena será incorrecto.

Para cumplir los objetivos ha sido necesario implementar un algoritmo de emparejamiento capaz de proporcionar un alto porcentaje de emparejamientos correctos, sin embargo en nuestro caso el tiempo de cálculo es medianamente elevado. Esto se debe a que al comparar las imágenes de la escena con las imágenes de los objetos de nuestra base de datos se han de realizar una cantidad elevada de operaciones dependiendo del número de imágenes de la escena y de los objetos, del numero de objetos de la base de datos, del número de puntos de interés de cada imagen, del tamaño del descriptor utilizado etc. Pese al conocimiento de que la utilización de un árbol kd-tree reduciría considerablemente el tiempo de cálculo, no ha sido posible su implementación debido a falta de tiempo. Por lo que si se desea utilizar la aplicación programada en este proyecto con una base de datos de tamaño mayor o que su tiempo de cálculo sea inferior sería recomendable considerar la modificación del algoritmo de emparejamiento para la utilización de arboles kd-tree. Para evitar tener que comparar todas las imágenes de la escena con todas las imágenes de la base de datos para ver si la escena tiene algún objeto de la base de datos se decidió añadir la posibilidad de comparar dos imágenes directamente y así evitar tiempo de cálculo.

7.2 Conclusiones Personales

Para finalizar considero conveniente realizar una serie de valoraciones personales sobre el trabajo dedicado a lo largo de este curso a este proyecto.

En primer lugar me ha parecido una experiencia muy interesante la realización de este proyecto. Entre los principales motivos a mencionar están que gracias a ello he conseguido adoptar las aptitudes necesarias para superar un problema complejo y diferente a los que me había enfrentado a lo largo de la carrera, adquiriendo conforme trabajaba conocimientos nuevos y desconocidos para mí.

La realización de esta memoria me ha ayudado también a darme cuenta de lo importante y difícil que puede ser plasmar por escrito una visión clara del fruto de trabajo realizado a lo largo de un proyecto.

El hecho de verme obligado a superar ciertos retos y obstáculos a lo largo del proyecto para conseguir avanzar me ha permitido enriquecerme profesionalmente y mejorar mi capacidad para afrontar dificultades.

En definitiva considero la realización de este proyecto como una experiencia positiva, enriquecedora y satisfactoria. Espero que este proyecto a su vez sirva como base para futuros proyectos que permitan mejorar y ampliar el trabajo aquí realizado.

8. Bibliografía

Papers

[1] *Noah Snavely, Steven M. Seitz, Richard Szeliski "Modeling the World from Internet Photo Collections". International Journal of Computer Vision, 2007*

[2] *Yasutaka Furukawa, Jean Ponce "Accurate, Dense and Robust Multi-View Stereopsis". IEEE Transactions on Pattern Analysis and Machine Intelligence, 2009*

[3] *Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool, "SURF: Speeded Up Robust Features". Computer Vision and Image Understanding (CVIU), Vol. 110, No. 3, pp. 346--359, 2008*

[4] *Lowe, David G. "Object recognition from local scale-invariant features". Proceedings of the International Conference on Computer Vision, 1999*

[5] *Noah Snavely, Steven M. Seitz, Richard Szeliski. "Photo Tourism: Exploring image collections in 3D". ACM Transactions on Graphics (Proceedings of SIGGRAPH 2006), 2006.*

[6] *Yasutaka Furukawa y Jean Ponce "Accurate, Dense and Robust Multi-View Stereopsis". IEEE Computer Society Conference on Computer Vision and Pattern Recognition, July 2007.*

[7] *A.M Romero y M.Cazorla* "Comparativa de detectores de características visuales y su aplicación al SLAM" *X Workshop de agentes físicos, Septiembre 2009.*

[8] *Yasutaka Furukawa, Brian Curless, Steven M. Seitz and Richard Szeliski* "Reconstrucing Bulding Interiors from Images" *ICCV 2009.*

[9] *Marc Pollefeys* "Visual modeling with a hand-held camera" *International Journal of Computer Vision 2004.*

[10] *Lowe, David G.* "Distinctive Image Features from Scale Invariant Features". *International Journal of Computer Vision, Vol. 60, No. 2, 2004*

[11] *J. Matas, O. Chum, M. Urban, T. Pajdla* "RobustWide Baseline Stereo from Maximally Stable Extremal Regions". Proc. of British Machine Vision Conference, pages 384-396, 2002.

Páginas Web

[12] http://phototour.cs.washington.edu/bundler/

[13] http://grail.cs.washington.edu/software/pmvs/

[14] http://www.vlfeat.org/

[15] http://www.wikipedia.org/