Proyecto Fin de Carrera
Ingeniería de Telecomunicación

Búsqueda de patrones para la mejora del proceso productivo y análisis de posicionamiento y profundidad

Autor: Martina Rivera Díez
Tutor: Fernando Caballero Benítez

Dep. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2015

Proyecto Fin de Carrera
Ingeniería de Telecomunicación

Búsqueda de patrones para la mejora del proceso productivo y análisis de posicionamiento y profundidad

Autor:
Martina Rivera Díez

Tutor:
Fernando Caballero Benítez

Dep. de Ingeniería Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2015

Proyecto Fin de Carrera: Búsqueda de patrones para la mejora del proceso productivo y análisis de posicionamiento y profundidad

Autor: Martina Rivera Díez

Tutor: Fernando Caballero Benítez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2015

El Secretario del Tribunal

Sobre el autor:

Martina Rivera Díez

Grado en Ingeniería de Tecnologías de Telecomunicación

Universidad de Sevilla *marrivdie@gmail.com*

La utopía está en el horizonte.
Me acerco dos pasos, ella se aleja dos pasos.
Camino diez pasos y el horizonte se desplaza
diez pasos más allá.
Por mucho que camine, nunca la alcanzaré.
Entonces, ¿para qué sirve la utopía?
Para eso: sirve para caminar.
- Eduardo Galeano-

A mi familia
y amigos.

Agradecimientos

Primeramente querría agradecer a la empresa FAICO la experiencia que gané trabajando con ellos y a mi tutor, Fernando, por su implicación y su ayuda para llevar este trabajo adelante. Gracias por la oportunidad que me habéis brindado y por todo lo que me habéis enseñado en este tiempo. Vosotros, junto al resto de profesores que me impartieron clase en esta escuela, me habéis dado los conocimientos y las bases que, a partir de ahora, me abrirán las puertas a un futuro.

En segundo lugar pero no por ello menos importante darle las gracias con todo mi corazón a mi familia, a la que adoro, en especial mis padres y mi hermano. Si he llegado hasta aquí, es porque vosotros siempre habéis estado ahí, y siempre habéis creído y confiado plenamente en mí y en mi capacidad para aprender y alcanzar las metas que me proponía, incluso cuando yo dejaba de hacerlo. El núcleo familiar que juntos hemos creado siempre ha sido elemental en el transcurso de este camino, y el apoyo incondicional y desinteresado que he recibido han servido para que no cayese en el intento. Quiero daros las gracias a mi padre, Antonio, y a mi madre, Carmen, por haberme brindado todos los medios necesarios para poder lograr mis retos y por asegurarse de que nunca nos faltase nada ni a Mario ni a mí. Nos enseñasteis desde pequeños el significado de las palabras sacrificio, esfuerzo y perseverancia. Fuisteis y seguiréis siendo un ejemplo a seguir. Vuestros propios éxitos fueron los que me han servido de ejemplo para entender cómo deben conseguirse las cosas. Me ayudasteis a centrarme y me orientasteis siempre hacia el camino correcto, respetando plenamente las decisiones que yo iba tomando poco a poco por mi cuenta. Creasteis un clima en casa que nos hizo amar nuestro hogar, a pesar de lo lejos que quisiésemos volar. Nunca perdimos la ilusión de volver. La comunicación con nosotros ha sido plena, y habéis compartido el papel de padres con el de amigos, a los que podíamos y siempre podremos acudir con cualquier tipo de percance o inquietud. Gracias por todo, os quiero.

Gracias a Sevilla y a tu compañía en estos últimos meses, contigo he probado el verbo disfrutar en todas sus posturas y sabores. Has llenado el cierre de esta etapa de ilusiones recogidas en un brindis por la próxima, en la que por supuesto sigo contando contigo, y lo has hecho maravillosamente. GRACIAS.

Gracias a BLAACASMIT y a aquellos amigos que han compartido conmigo este bonito camino desde los primeros días, en especial a Glou, Pelocho, Rafa y Rober. También a aquellos que fui conociendo por el camino y que se volvieron igual de importantes para mí: Helena, Luis, María y Joseda. Gracias a todos porque sin vosotros no hubiese sido tan fácil guardar un buen recuerdo de estos años. Siempre tendré un hueco para vosotros en mi corazón.

Andrés, Helena, Irene y María José, os volvisteis esenciales para mí y me ayudasteis siempre que lo necesite. Os estaré eternamente agradecida y una parte de este trabajo os pertenece a vosotros.

Gracias a mis viejos amigos, los de toda la vida. Aquellos que han hecho de la distancia omiso caso y han conseguido fortalecer nuestra amistad cada día un poco más. A mi pandilla, no tengo nada que deciros porque de mi todo ya sabéis. Por siempre TGON.

Por último hacer una mención especial a mi abuelo Antonio, que no estará presente para compartir conmigo este gran paso, pero que siempre creyó en mí y me dio un refugio en el que resguardarme en esa etapa tan intensa que es la adolescencia, en la que los cambios a los que me enfrentaba a veces podían conmigo. Gracias abuelo. Formarás siempre parte de lo que soy y de lo que llegue a ser porque tu ausencia nunca podrá convertirse en olvido.

Martina, Septiembre de 2015.

Resumen

Este proyecto consiste en el diseño de una aplicación de un sistema de visión artificial que seleccione los patrones de mayor correspondencia con una imagen y que sea capaz de estimar las distancias a las que se encuentran algunos de los objetos de interés y conocer su localización, traslación y rotación.

Recientemente han aparecido dispositivos que permiten obtener información tridimensional de una escena. Uno de los más utilizados es el constituido por un par estereoscópico de cámaras. El objetivo de este proyecto es obtener coordenadas 3D de una escena real con el procesamiento de una sola cámara haciendo uso de la técnica de 'matching', que consiste en la búsqueda de correspondencias de color y forma entre objetos, extrayendo sus puntos característicos. En este caso será entre los patrones de la base de datos y la imagen capturada en cada instante. El software y algoritmos están diseñados en C++, utilizan las librerías de visión OpenCV® y están soportados por Windows®.

La función principal del programa es el cálculo de la traslación, rotación, escalado y coordenadas de profundidad o coordenadas 3D, salvo un factor de escala, en tiempo real de un objeto. También se necesita minimizar los tiempos de ejecución, buscar los algoritmos óptimos y obtener flexibilidad en cuanto a patrones modificables o ampliaciones futuras sobre la base de datos con la que se quiera trabajar. Los pasos que hemos seguido para alcanzar cada objetivo y los recursos que se han implementado son:

- Creación de una Base de Datos con los patrones que queremos identificar.
- Obtención del vector descriptor para los puntos característicos de cada patrón mediante el descriptor y detector del algoritmo SURF.
- Bucle de capturas continuas de la cámara o vídeo precargado, aplicando a cada una de ellas el paso anterior.
- Comparación de descriptores generados en cada captura con el vector descriptor calculado en la segunda fase y selección de aquel patrón de mayor correspondencia con la imagen capturada en cada instante.
- Cálculo del contorno y localización del patrón identificado en la captura.
- Obtención de los datos de interés (localización, traslación, rotación y mapa de profundidad).

Para evitar errores, en los cambios bruscos de iluminación o en situaciones con exceso de ruido, se ha elegido el algoritmo SURF, "Speeded-Up Robust Features". SURF es menos sensible a cambios que su antecesor SIFT, "Scale-invariant feature transform". También se ha reducido el número de puntos característicos que se obtenían en cada proceso, filtrando sólo aquellos que cumpliesen unas distancias mínimas determinadas. De esa manera conseguimos menos carga computacional y se trabajará solamente con los puntos más representativos de cada imagen, mejorando el rendimiento de la aplicación.

Abstract

This work deals with the designing an implementing an artificial vision system that is capable of estimating the distances to which some objects of interest are located and to know their localization, translation and rotation.

Recently, devices that allow obtaining three-dimensional information from the scenes have been developed. One of the most popular techniques is stereoscopic vision, composed by a couple of synchronized cameras. This work presents an artificial vision approach for computing 3D coordinates of a real scene with only one camera. The software is implemented in C++ and the algorithms are designed using OpenCV computer vision libraries and are supported by Windows®. The phases we have taken to achieve each objective and the resources that have been implemented are:

- Creation of a database with models who we want to identify.
- Obtaining the descriptor vector for the characteristic points of each model by the descriptor and the detector of the SURF algorithm.
- Continuous loop from camera capture video or video preloaded, applying to each capture the previous step.
- Comparison of descriptor generated in the capture with the descriptor vector from the second step and selection of the model with more similitude to the image captured at every moment.
- Calculation of contour and localization of the model identified inside of the capture.
- Obtaining the details (location, translation, rotation and depth map with the respective 3D coordinates).

Índice

Agradecimientos **iii**

Resumen **iv**

Abstract **vii**

Índice **viii**

Índice de Figuras **x**

Notación **xiii**

1 Objetivo y Motivación del Trabajo **16**

1.1. Descripción del Trabajo Realizado *17*

1.2. Organización del Documento *18*

2 Estado del Arte **20**

2.1. Antecedentes *20*

2.1.1 Orígenes y Primeros Estudios de la Visión Artificial 20

2.1.2 Distintios enfoques dentro de la investigación 20

2.1.3 Estudios más Influyentes y Aparición de las Primeras técnicas para el Procesamiento de Imágenes Digitales 21

2.2. Algoritmos de Visión Artificial para el Reconocimiento de Patrones *21*

2.2.1 Algoritmo "Harris Corner Detection" 21

2.2.2 Algoritmo "Shi-Tomasi Corner Detector" 23

2.2.3 SIFT 23

2.2.3.1 Diferencia Gaussiana 24

2.2.3.2 Puntos Característicos 24

2.2.3.3 Orientación 24

2.2.3.4 Descriptor y Mapeo 24

2.2.4 SURF 24

2.3. Aplicaciones de la Visión Artificial en la Industria *25*

3 Solución Propuesta **29**

3.1. SURF (Speed Up Robust Feature) *29*

3.1.1. Historia 29

3.1.2. Algoritmo 29

3.1.2.1. Detección de Puntos de Interés 30

3.1.2.2. Asignación de la Orientación 32

3.1.2.3. Descriptor SURF 33

3.2. RANSAC (RANdom SAmple Consensus) *34*

3.2.1. Historia 35

3.2.2. Algoritmo 35

4 Detalles de Implementación 39
4.1. Introducción 39
4.2. Reconocimiento de Patrones en una Base de Datos 41
4.3. Búsqueda y Clasificación de un Patrón 42
4.3.1. Búsqueda de la Máxima Similitud 43
4.3.2. Algoritmos de Optimización 44
4.4. Cálculo del Contorno y Posicionamiento Dentro de la Escena 44
4.4.1. Cálculo del Contorno 44
4.4.1.1. Detección de Esquinas 44
4.4.1.2. Formación de un Contorno 44
4.4.2. Obtención de Características de Posicionamiento 45
4.5. Funciones Externas al Programa Principal 46
4.5.1. getImagesFromDirectory 46
4.5.2. Inicializar 46
4.5.3. Pendiente 47
4.5.4. ExtracTransH 47

5 Simulaciones y Análisis de los resultados 50
5.1. Lenguaje de Programación 50
5.2. Estructura del Programa 50
5.3. Pruebas de Calidad y Desempeño del Sistema 52
5.3.1. SIFT vs. SURF 53
5.3.2. Algoritmos de Optimización 54
5.3.3. Cálculos del Contorno 55

6 Conclusiones y Trabajos Futuros 58
6.1 Cumplimiento de Objetivos y Conclusiones 58
6.2. Líneas Futuras 59

Referencias 63

Índice de Figuras

1. Objetivo y Motivación de Trabajo

Figura 1-1. Pasos iniciales del proceso realizado en la fase de extracción de características. 17

Figura 1-2. Pasos finales del proceso de extracción de características de una imagen. 18

2. Estado del Arte

Figura 2-1. Ecuación para la detección de esquinas por el método de *Harris*. 21

Figura 2-2. Imagen esquemática del algoritmo Harris. 22

Figura 2-3. Resultado de una detección de esquinas con el algoritmo Harris. 22

Figura 2-4. Imagen esquemática del algoritmo Shi-Tomasi. 23

Figura 2-5. Resultado de una detección de esquinas con el algoritmo Shi-Tomasi. 23

Figura 2-6. Ampliación de la esquina enventanada. 23

Figura 2-7. Espacio escala mediante SIFT (izq) y SURF (dcha). 25

Figura 2-8. Cámara inteligente durante un proceso de embotellado. 26

3. Solución Propuesta y Descripción de la Teoría despues de la solución

Figura 3-1 Esquema del proceso para realizar nuestro proyecto de clasificación SURF 30

Figura 3-2 Representación de la derivada parcial de segundo orden de un filtro gaussiano 31

Figura 3-3 Escala para el descriptor SURF 32

Figura 3-4 Cálculo de la respuesta Haar 32

Figura 3-5 Cálculo direcciones x e y 33

Figura 3-6 Descriptores SURF 34

Figura 3-7 RANSAC. Puntos fuera y dentro del Umbral. 37

4. Descripción de la solución adoptada

Figura 4-1. Organigrama de los procesos del programa(I). 39

Figura 4-2. Organigrama de los procesos del programa(II). 40

Figura 4-3. Ejemplo 1 de una intervención de BFMatcher. 41

Figura 4-4. Ejemplo 2 de una intervención de BFMatcher. 41

Figura 4-5. Ejemplo 1 de cálculo del contorno adaptativo a distintas orientaciones. 44

Figura 4-6. Ejemplo 2 de cálculo del contorno adaptativo a distintas orientaciones. 45

5. Simulcaciones y Análisis de los Resultados

Figura 5-1: Menú de Inicio del Programa 50

Figura 5-2: Detección de fases e instrucciones a seguir. 50

Figura 5-3: Matriz de componentes de profundidad 3D. 51

Figura 5-4: Información de la escala, traslación y rotación del objeto localizado dentro de la escena. 51

Figura 5-5: Falso positivo (Ejemplo A). 52

Figura 5-6: Falso positivo (Ejemplo B). 52

Figura 5-7: Falso negativo 52

Figura 5-8: Obtención de puntos característicos detectados por el algoritmo SIFT. 53

Figura 5-9: Obtención de puntos característicos detectados por el algoritmo SURF. 54

Figura 5-10: Error de contorno. Ejemplo A 55

Figura 5-11: Error de contorno. Ejemplo B 55

Figura 5-12: Error de contorno. Ejemplo C 55

6. Conclusiones y Líneas Futuras

Figura 6-1. Captura del proceso de montaje en realidad aumentada desarrollado por la compañía BWM 59

Figura 6-2. Ejemplo de implementación de la Visión Artificial en el campo de la medicina. 59

Figura 6-3. Ejemplo de implementación de la Visión Artificial en el teléfono móvil como complemento de un programa de uso del GPS. 59

Figura 6-4. Ejemplo de implementación de la Visión Artificial en el campo de los videojuegos. 60

Figura 6-5. Ejemplo de implementación de la Visión Artificial en la domótica o el uso cotidiano del usuario. 60

Notación

>	Mayor o igual
>>	Mucho mayor que
:	Tal que
<	Menor o igual
Cos	Función coseno
Sen	Función seno
sign	Símbolo
Tan	Función tangente
atan	Función arco tangente
~	Aproximación
I_x	Derivada de la imagen en la dirección x
I_y	Derivada de la imagen en la dirección y
det	Determinante
min	Mínimo
λ	Autovalor
log	Logaritmo en base 10
√	Raíz cuadrada
det.	Determinante
∑	Sumatorio

1 OBJETIVO Y MOTIVACIÓN DEL TRABAJO

Para obtener, hay que otorgar antes, para ganar, hay que perder un poco.

- Carlos Salem -

Este proyecto, titulado "Búsqueda de patrones para mejora del proceso productivo y análisis de posicionamiento y profundidad", tiene como objetivo el diseño y la implementación de un sistema de visión estéreo que permita posicionar de forma autónoma objetos (ej.: productos alimenticios, piezas de montaje, etc.) que se encuentren presentes en la escena capturada por la cámara en el estudio. Con este trabajo hemos tratado de unir en un mismo sistema lo que se ha venido haciendo años atrás sobre mapas de profundidad [1], [2] pero con algunas características presentes en los trabajos sobre estimación de distancias mediante visión estéreo [3].

El área de Visión Artificial se encuentra en plena vía de desarrollo. La motivación en el estudio de este área viene fomentada por la posibilidad de llegar a sustituir a la visión humana en ciertas tareas (siendo más rápido, más correcto y a buen precio). Es una técnica muy compleja que cuenta con cientos de aplicaciones que permiten a los ordenadores ver y entender su entorno. Está experimentando un espectacular auge y las compañías están gastando una cantidad, cada vez mayor, de dinero para ayudar a los dispositivos electrónicos a reconocer imágenes, al igual que pueden hacerlo los seres humanos. Ahora las empresas pueden comprar cámaras cuatro veces más pequeñas para capturar y procesar imágenes de alta calidad y hace apenas tres años esto no habría sido posible. Los científicos e ingenieros de la visión por computador y reconocimiento de patrones están creando un mundo en el que los coches se conducen solos, las máquinas reconocen a las personas y logran "entender" sus emociones, y los robots humanoides viajan sin vigilancia, realizando todo tipo de funciones, desde tareas en fábricas hasta rescates de emergencia.

Consideramos un paso crucial, para las próximas generaciones, la creciente evolución y poder adquirido de la visión por computador en sistemas de inteligencia robótica y artificial. Una vez que las máquinas puedan identificar objetos y comprender su entorno, serán capaces de extender el alcance de los seres humanos o de sustituirlos. Estudios como los del libro "Inteligencia Artificial, el futuro del hombre." de Alejandro Madruga González nos muestran los campos más propensos a experimentar los mayores avances en esta tecnología, que se encuentra en pleno crecimiento y evolución. Serán, entre otros:

- **Redes neuronales:** Sus investigaciones comienzan a centrarse en el órgano más importante de nuestro cuerpo, el cerebro y sus conexiones internas.

- **Máquinas súper-inteligentes:** Se cifra la esperanza en el hardware con máquinas cada vez más potentes (ley de Moore), que con el tiempo llegarán a superar la potencia del cerebro, en lugar de aspirar a descifrar la mente por medio del software.

- **Algoritmos genéticos:** Se aceptan los avances de otras ciencias, como la biología, en lugar de atrincherarse en modelos exclusivamente informáticos.

- **Robots reactivos**, se desarrollan robots de tamaños reducidos capaces de interactuar con el medio y desarrollar un aprendizaje mediante los obstáculos que se vaya encontrando, en lugar de los sistemas basados en el conocimiento, que están desligados de la realidad y hay que alimentarlos de forma manual, de ahí que recibieran el nombre de sistemas autistas.

En este proyecto se aborda el análisis para la clasificación y búsqueda automática de objetos de forma rápida, sobre una base de datos arbitrariamente grande. El sistema es capaz de estimar las distancias a las que se encuentran algunos de los objetos de interés y conocer información acerca de su localización, su traslación, su rotación y su mapa de profundidad.

1.1 Descripción del trabajo realizado

En primer lugar se realiza un montaje con una Webcam y se define una Base de Datos con la que vamos a trabajar. Se obtienen capturas a tiempo real desde la cámara, en las que se buscan los patrones almacenados en la base de datos. El umbral utilizado por nuestro descriptor-extractor SURF define el requisito necesario de cercanía que deben de presentar los puntos extraídos de las capturas para, a su vez, ser puntos característicos de la imagen.

Tras ello se calcula la detección y descripción de los puntos característicos. En la figura 1-1 se muestran de forma esquemática las tres primeras fases antes explicadas:

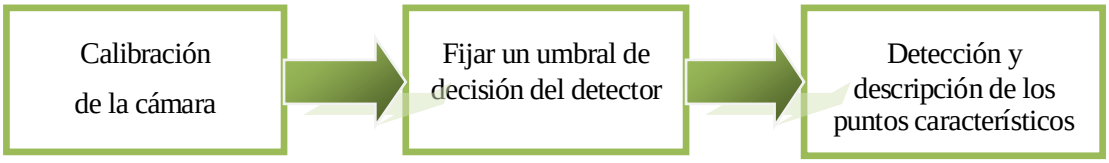


Figura 1-1. Pasos iniciales del proceso realizado en la fase de extracción de características.

Posteriormente, una vez aplicado el detector para extraer los puntos característicos de cada uno de nuestros patrones de la imagen capturada, se realizan los siguientes pasos fundamentales:

- Asociación de los puntos de interés localizados: comparación de descriptores generados en cada captura con el vector descriptor calculado en cada patrón de la bases de datos.
- Selección del patrón de mayor correspondencia con la imagen capturada en cada instante.
- Algoritmos de optimización, dónde se exigen unos requisitos mínimos para la identificación correcta del patrón dentro de la imagen. Aquí se calcula posteriormente el contorno y localización del patrón identificado en la captura, así como la obtención de los datos de interés.

Figura 1-2*:* Pasos finales del proceso de extracción de características de una imagen.

Tras identificar el patrón o modelo situado en la escena se calcula, mediante distintos algoritmos matemáticos (que explicamos con detenimiento en el apartado 4.3.2), la información de interés que necesitamos: datos sobre la estimación de distancias, localización, traslación, rotación, escala y su mapa de profundidad.

A continuación se realizan una serie de grabaciones y toma de imágenes en distintas partes del proceso, con el sistema en movimiento o parado, en diferentes condiciones de luz, etc. Con esas imágenes se efectúan numerosas pruebas que permiten obtener mapas de disparidad (con los puntos característicos y distancias entre ellas).

Finalmente, tras analizar todos los resultados y optimizar las soluciones con distintos procedimientos, se ajusta todo el sistema hasta lograr los objetivos propuestos.

1.2 Organización del documento

Este proyecto se ha dividido en varias partes para facilitar la comprensión y el desarrollo realizado.

El **capítulo 2** incluye, bajo el epígrafe "Estado del Arte", los inicios del estudio de esta materia, los primeros nombres influyentes e investigaciones relevantes, así como su evolución a lo largo del tiempo. Se citan los problemas que se encuentran a medida que avanzan y crecen los objetivos, así como los progresos y las soluciones que se descubren a lo largo de los años. En este capítulo también se habla de los algoritmos actuales con los que trabaja la visión artificial y las aplicaciones de esta tecnología en el campo de la industria.

En el **capítulo 3** se describen los métodos usados en el programa desde un punto de vista teórico. Se profundiza en las bases matemáticas de los algoritmos que se han seleccionado así como en sus características principales haciendo hincapié en el extractor de características SURF y en el algoritmo de estimación RANSAC.

El **capítulo 4** describe el código en las fases principales del programa, explicando con el mayor detalle posible las funciones que se han implementado para su funcionamiento.

En el **capítulo 5** se exponen los resultados y las pruebas realizadas durante el tiempo de trabajo. Se analizan algunos datos y resultados obtenidos, tiempos de ejecución y causas o problemas por los cuales se han producido algunos errores a lo largo del proceso, así como las soluciones adoptadas.

El **capítulo 6** muestra las conclusiones del estudio, aportando ideas sobre lo que podrían ser las futuras líneas de trabajo y las aplicaciones que podrían desarrollarse a partir de este proyecto.

2 ESTADO DEL ARTE

El bien es lento porque va cuesta arriba.

El mal es rápido porque va cuesta abajo.

- Alejandro Dumas -

La visión artificial, también conocida como visión por computador (del inglés *computer vision*) o visión técnica, es un subcampo de la inteligencia artificial. El propósito de la visión artificial es programar un ordenador para que sea capaz de interpretar una escena, obtener las características de una imagen y, posteriormente, información sobre dicha imagen.

En este proyecto se realiza un estudio sobre una de sus muchas vertientes y campos de aplicación. Es sabido que una investigación no puede ser concebida como un "acto", sino que es un "proceso" que implica continuidad en cada una de sus fases. Cada paso es útil para la construcción del siguiente, pero el proceso solo nada dice sino es en conexión con el trabajo realizado anteriormente por el colectivo de investigadores. Este capítulo se centra en los inicios de este campo de la tecnología, su origen y evolución a lo largo del tiempo. También se hace una breve mención a las tecnologías más relevantes y los últimos avances que se han obtenido hasta hoy.

2.1 Antecedentes

2.1.1 Orígenes y primeros estudios de la visión artificial

El interés por el estudio de la visión tuvo sus orígenes en la etapa griega. De ahí hasta nuestros días, se han formulado teorías sobre la percepción del ser humano de su realidad exterior y el uso de la información captada por sus sentidos.

Son los estudios de Johannes Kepler (1571-1630) los más cercanos de la historia sobre el análisis geométrico de la formación de la imagen en el ojo [4], y tras ellos los estudios de Isaac newton (1643-1727) sobre visión en color [5].

Ya a principios del siglo pasado, los estudios de Hermann Ludwig Ferdinand Von Helmholtz (1821-1894) sobre la óptica fisiológica [6] y Max Wertheimer (1880-1943) sobre el movimiento aparente de agrupaciones de puntos o campos [7][8], entre otros, establecieron las bases de las actuales teorías de la percepción visual. Desde la aparición de los primeros computadores digitales en los años setenta se puso de manifiesto su gran capacidad para tratar la información espacial, con aplicación directa en campos relacionados con el estudio de propiedades del sistema de visión humana.

2.1.2 Distintos enfoques dentro de la investigación

Debido a la gran complejidad del sistema de visión humana, el avance en las teorías y algoritmos que explicaban su funcionamiento y propiedades era lento y se buscaron formas más directas para abordar estos problemas a partir de tres enfoques distintos:

El primero se basó en criterios de tipo matemático, desarrollando técnicas empíricas que realizaban la estimación de bordes y líneas usando distintos métodos. En este primer grupo destaca Azriel Rosenfeld, con ideas interesantes como el uso simultáneo de operadores de distinto tamaño [9], pero este desarrollo no logra proponer métodos para evaluar los distintos algoritmos.

La segunda forma de enfocar estos problemas se basó en reducir su alcance a bloques blancos iluminados sobre fondo negro. Este modelo logró funcionar de manera perfecta para ciertos ensayos como los realizados por Waltz (1975) y Mackworth (1973) [10]. Ellos consiguieron resolver interpretaciones de dibujos lineales a raíz de estudios de imágenes de prismas sólidos. La simplificación tenía la finalidad de extrapolar los conocimientos adquiridos a casos más complejos.

Por último**, el tercer** enfoque se basó en modelos que trataron de completar la percepción visual y eliminar las limitaciones de los casos anteriormente citados. Destacan los trabajos realizados por Horn (1975-1977) sobre la formación de la imagen. Horn establece modelos de cálculo a través de ecuaciones diferenciales, para expresar la formación de la imagen, relacionando los valores de intensidad de dicha imagen con la geometría de la superficie [11] [12]. Se estudia con gran detalle el modo en que la iluminación, la geometría, la reflectancia de la superficie y el punto de vista del observador actúan de forma conjunta para crear los valores de intensidad medidos en la imagen.

2.1.3 Estudios más influyentes y aparición de las primeras técnicas para el procesamiento de imágenes digitales

Tras los modelos de Horn, diversos investigadores, entre los que destaca Julesz (1975), demuestran que los mecanismos de la visión estereoscópica son realizados en la retina en una etapa muy temprana del proceso de visión, y que el mecanismo de visión humana tiene la posibilidad de interpretar imágenes en 3D usando solamente las informaciones sobre profundidad, distancia y textura [13].

Posteriormente, Marr establece las bases en el campo del estudio de los mecanismos de la visión humana y el análisis de imágenes digitales [14], consiguiendo realizar métodos de descomposición de una imagen para niveles 2D y 3D.

De esta forma comienza la aparición de técnicas asociadas a la codificación, captura y representación de las imágenes, técnicas para el análisis de imágenes digitales, técnicas de visión por una computadora o un robot, con el objetivo de extraer información presente en una imagen para lograr interpretar la escena representada por dicha imagen.

2.2 Algoritmos de Visión Artificial para el Reconocimiento de Patrones

Los algoritmos relacionados con visión artificial y el reconocimiento de patrones son muy variados y abarcan numerosas técnicas y objetivos. Así, en este capítulo se van a citar y explicar brevemente los diferentes algoritmos que existen en OpenCV para encontrar características dentro de una imagen, usando como referencia los tutoriales del propio desarrollador [15].

2.2.1 Algoritmo “Harris Corner Detection”

Sabemos que las esquinas son regiones dentro de una imagen que contienen una gran variación en la intensidad, en todas las direcciones. El primer intento por conseguir detectar estas esquinas vino de la mano de Chris Harris y Mike Stephens (1988), que dio nombre al detector “Harris Corner Detector”. Se basa en una idea matemática sencilla, donde se busca la diferencia entre las intensidades que se generan en un desplazamiento de (u, v) en todas sus direcciones [16].

$$E(u,v) = \sum_{x,y} \underbrace{w(x,y)}_{\text{window function}} \; [\underbrace{I(x+u,y+v)}_{\text{shifted intensity}} - \underbrace{I(x,y)}_{\text{intensity}}]^2$$

Figura 2-1. Ecuación para la detección de esquinas por el método de *Harris*.

Para la detección de las esquinas se debe maximizar E (u,v). Por ello, aplicando Taylor a la ecuación anterior obtenemos la ecuación final como:

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$

Dónde:

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix}$$

Aquí, I_x e I_y son las derivadas de la imagen en las direcciones x e y, respectivamente.

Tras esto, viene la parte principal del algoritmo donde ellos crean una ecuación, que determinará si una ventana puede contener una esquina o no:

$$R = \det(M) - k(trace(M))^2$$

Dónde:

- $\det(M) = \lambda_1 * \lambda_2$
- $trace(M) = \lambda_1 + \lambda_2$
- λ_1 y λ_2 son los valores eigen[1] de M

Con los valores de λ_1 y λ_2 de M podremos, decidir si una región es esquina, borde o plana:

- Cuando $|R|$ es pequeño, lo que sucede cuando λ_1 y λ_2 son pequeñas, la región es plana.
- Cuando R< 0, lo que sucede cuando $\lambda_1 \gg \lambda_2$, o viceversa, la región es borde
- Cuando R es grande, lo que sucede cuando λ_1 y λ_2 son grandes y $\lambda_1 \sim \lambda_1$, la región es una esquina.

[1] *Palabra alemana con significado equivalente en este contexto a valor propio, autovalor o valor característico*

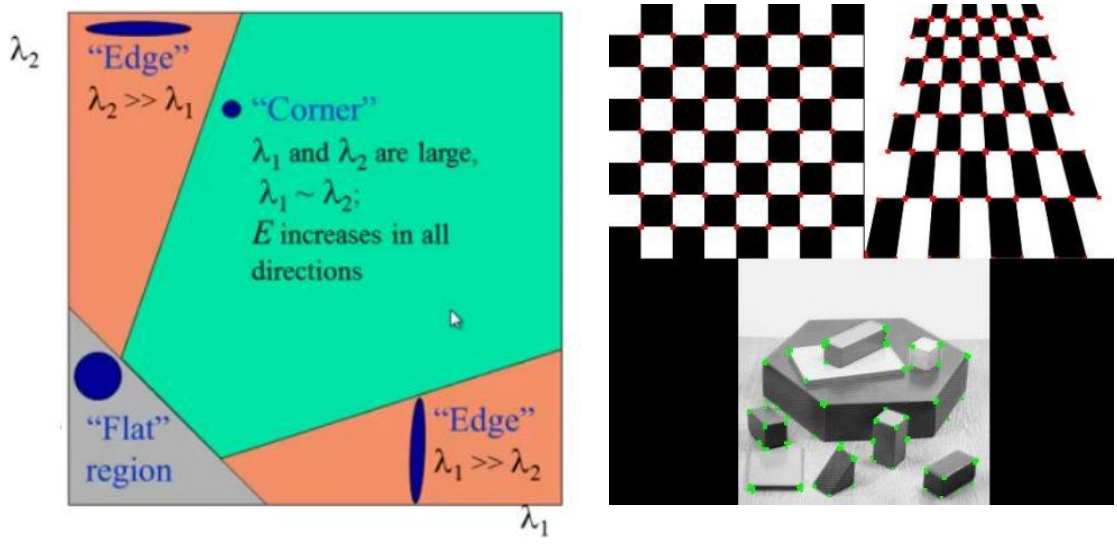


Figura 2-2 (izq.) Imagen esquemática del algoritmo Harris. [15]

Figura 2-3(dcha.) Resultado de una detección de esquinas con el algoritmo Harris. [15]

2.2.2 Algoritmo "Shi-Tomasi Corner Detector"

En 1994 J. Shi y C. Tomasi realizan pequeñas modificaciones y consiguen mejores resultados, en comparación con Harris Corner Detector. Este algoritmo propone los siguientes cambios respecto al anterior:

$$R = \min(\lambda_1 \lambda_2)$$

Si el valor que obtenemos es mayor al valor umbral, nos encontraremos en una esquina. Si trazamos de nuevo el dibujo esquemático, nos encontramos con la siguiente figura, en la que se puede ver a simple vista que solamente se considera esquina cuando los valores λ_1 y λ_2 se encuentran ambos por encima de λ_{min} :

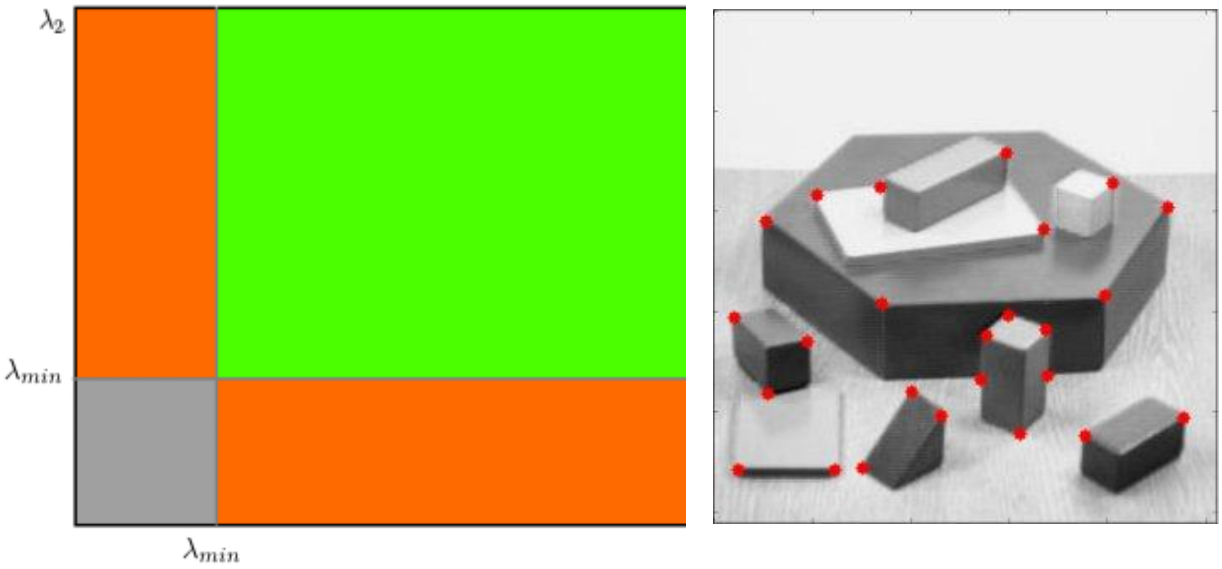


Figura 2-4 (izq.) Imagen esquemática del algoritmo Shi-Tomasi.

Figura 2-5(dcha.) Resultado de una detección de esquinas con el algoritmo Shi-Tomasi.

2.2.3 SIFT (“Scale-invariant feature transform”)

Los resultados que se obtienen con los métodos anteriores no son invariantes a los cambios de escala. Reconocemos una esquina pequeña dentro de una escena, con un enventanado de cierta escala. Si aumentamos la imagen de la escena, la ventana considerará dicha esquina como un contorno plano.

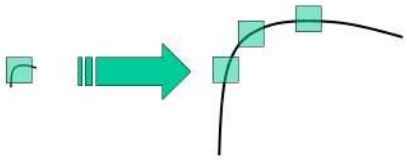


Figura 2-6 Ampliación de la escena enventanada.

Es en el 2004 cuando aparece el algoritmo SIFT, que viene a dar solución a este problema. Consta de cuatro etapas: generación del espacio escala mediante funciones diferencia de gaussianas (DoGs); localización de puntos invariantes (máximos y mínimos del espacio escala); asignación de orientación a los puntos (en base a direcciones locales de los gradientes) y generación del descriptor del punto. Comentamos brevemente cada una de ellas:

2.2.3.1 Diferencia Gaussiana

Se consigue con las diferencias obtenidas de los desenfoques Gaussianos de una imagen. Aquí se calcula la escala en la que estará mejor representado cada punto clave identificado en nuestra imagen. Se resuelven los problemas encontrados en algoritmos anteriores.

2.2.3.2 Puntos característicos

Una vez que se ubican los puntos clave de la imagen, se aplica el desarrollo en serie de Taylor para la ubicación exacta de los extremos y si la intensidad en dichos extremos es menor que el valor umbral, estos puntos serán descartados. Los bordes también presentan una fuerte relación de contraste, por lo que deberemos eliminar cualquier punto clave de bajo contraste y quedarnos con aquellos puntos de interés que presenten fuertes diferencias de contraste. La precisión obtenida en esta fase será decisiva para la buena aproximación de nuestros resultados.

2.2.3.3 Orientación

Con la orientación de cada punto clave obtenido se garantiza que la rotación de dicha imagen no modifique los puntos clave. Se tomará una vecindad alrededor de cada punto significativo, acorde con la escala y magnitud del gradiente. Con dicha vecindad se calcula la dirección de su región.

Se crea un histograma de gradientes orientados, creando puntos significativos con la misma ubicación y escala, pero con diferentes direcciones. De esta forma se consigue más estabilidad.

2.2.3.4 Descriptor y Mapeo

Es la última parte y la más importante del algoritmo, dónde se construyen los descriptores y se identifican los puntos comunes de dos imágenes (Keypoint Matching). Se crea el descriptor de cada punto característico tomando una vecindad de cierta escala alrededor de éste, repetidas veces, dividiéndose en sub-bloques para formar el descriptor en dicho punto. Así, logramos robustez frente a cambios de iluminación, giro

y rotación. Los puntos clave entre dos imágenes se corresponden con la identificación de sus vecinos más cercanos. De este algoritmo, lento y robusto en principio, se hizo necesaria, posteriormente, una versión más acelerada.

2.2.4 SURF ("Speeded-Up Robust Features")

El algoritmo SURF viene del acrónimo en inglés "Características robustas aceleradas". Fue en el 2006 cuando tres personas, Bahía H, Tuytelaars T, y Van Gool L, publicaron este algoritmo mejorado que, como su nombre indica, es una versión acelerada de SIFT. La cuatro etapas del algoritmo son: generación de imagen integral para agilizar los cálculos; creación del espacio escala mediante aproximaciones a la segunda derivada de la gaussiana; localización de puntos invariantes (máximos y mínimos del espacio escala); asignación de orientación (con wavelets Haar) y generación del descriptor.

Siguen confiando en la matriz gaussiana para el escalado y la ubicación de los puntos característicos pero para la asignación de la orientación, SURF utiliza las respuestas wavelt[2] de dirección horizontal y vertical de una vecindad. La respuesta de ondas se puede llevar a cabo fácilmente con imágenes integrales de cualquier escala, permitiendo la rotación. De esta forma, ya no se requiere tener conocimiento de la orientación de cada punto característico y se consigue acelerar mucho el proceso.

Otra mejora interesante es el uso de la señal de Laplace que, sin añadir dificultad computacional (porque ya se calcula durante la detección), logra distinguir con el signo del Laplaciano manchas brillantes en fondos oscuros y a la inversa, manchas oscuras en fondos brillantes. Para la etapa de adaptación, donde SIFT localizaba los puntos característicos de la imagen, esta información permite una ejecución más rápida sin reducir el rendimiento del descriptor.

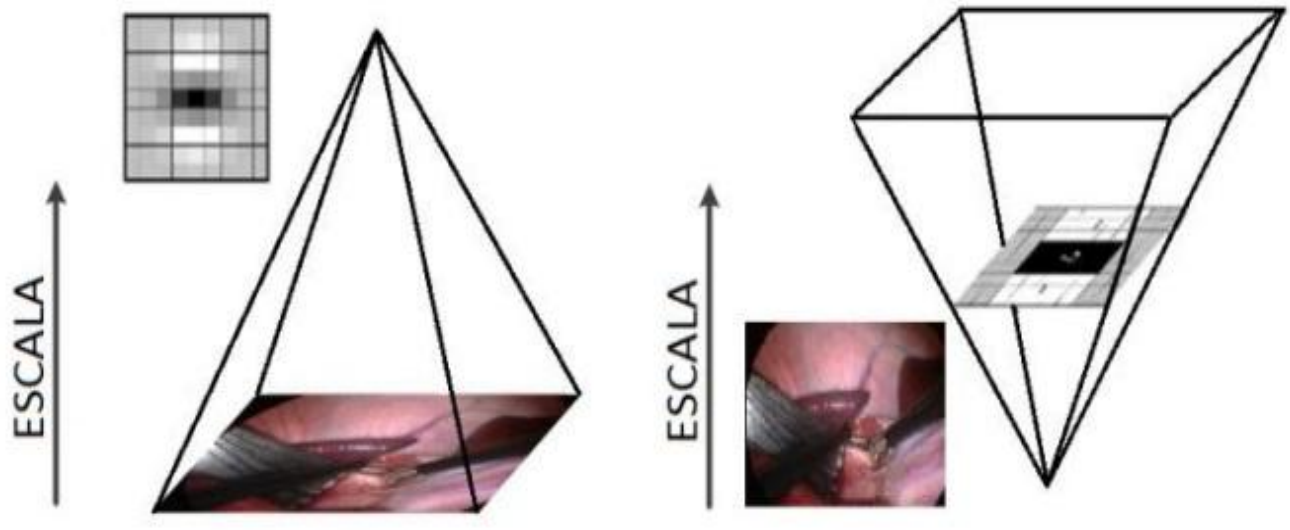


Figura 2-7. Espacio escala mediante SIFT (izq) y SURF (dcha). [19]

2.3. Aplicaciones de la Visión Artificial en la Industria

La visión artificial aplicada a la industria abarca ramas muy diferentes, como son informática, óptica, ingeniería mecánica, ingeniería de imagen, automatización industrial...

---

[2] *La transformada de ondícula, también transformada wavelet, es un tipo especial de transformada matemática que representa una señal en términos de versiones trasladadas y dilatadas de una onda finita (denominada óndula madre).*

A diferencia de las aplicaciones en el mundo académico, centradas principalmente en el procesamiento de imágenes, las aplicaciones de visión artificial a la industria integran dispositivos de entrada/salida, sistemas de captura de imágenes digitales ó equipos con fines de fabricación de sistemas robóticos. Los sistemas de visión artificial realizan también inspecciones visuales que requieren alto rendimiento y funcionamiento continuo o repetitivo de las medidas.

Sus objetivos suele centrarse en la comprobación del cumplimiento de ciertos requisitos para una pieza, tales como las dimensiones, color y forma, correcto etiquetado, números de serie, la presencia de componentes, etc. Como ejemplos más populares podemos mencionar:

- **Industrias de la Alimentación:** su adaptación fue más compleja debido a la diversidad de formas que se presentan en los productos naturales como frutas, hortalizas, pescado… La introducción de la visión artificial en este sector ha optimizado de forma notable los procesos de producción.
- **Industria de Automoción:** los altos controles de calidad y automatización industrial que deben pasar los productos del sector del automovilismo y los grandes presupuestos que se invierten en este sector han supuesto una gran aceptación de esta tecnología por parte de las industrias, siendo uno de los mayores consumidores.
- **Packaging:** el envasado de bebidas, donde existe alto nivel de competitividad entre empresas, crea la necesidad de conseguir gran velocidad de producción y altos niveles de calidad.

Figura 2-8 Cámara inteligente durante un proceso de embotellado. [20]

- **Electrónica:** la industria de la electrónica es conocida por el elevado ritmo de producción y automatización de sus procesos. La visión artificial se aplica desde la fase de manipulación e identificación de componentes hasta la fase del control de calidad.
- **Industria Farmacéutica:** otra de las industrias que más dinero mueve y que invierte en esta tecnología para sus procesos productivos desde hace tiempo. La responsabilidad que conlleva la distribución de medicamentos y las dosis correctas que debe contener cada envase suministrado hace necesarios los controles de calidad y seguridad que se facilitan por medio de robots y sistemas de visión automatizados.
- **Otras industrias importantes:** Empresas de seguridad y video-vigilancia, regulación del tráfico y tránsito vial, industria textil…

3 SOLUCIÓN PROPUESTA

La cura para todo es el agua salada:
sudor, lágrimas o el mar.
- Isak Dinesen -

El extractor de características SURF se trata de una técnica de obtención de los de puntos de interés de una imagen. Mediante SURF se consigue un conjunto de descriptores muy distintivos e invariantes ante cambios en escala, iluminación y perspectiva, para cada uno de los puntos seleccionados de la imagen. Por ello, el emparejamiento de estos puntos entre dos imágenes distintas es muy robusto. El coste computacional que presenta este extractor será inferior, en comparación con otros similares como su antecesor SIFT.

SURF realiza la extracción de los puntos detectando en primer lugar los posibles puntos de interés y su localización dentro de una imagen. Tiene un tiempo de procesamiento más rápido que el método SIFT, ya que los keypoints contienen un número menor de descriptores debido a que la mayoría de los descriptores son 0. Este descriptor se puede considerar una mejora del descriptor SIFT ya que mejora su funcionamiento utilizando la gran mayoría de las funciones anteriores. Del trabajo de Aracil López, Rafael. "Desarrollo de un sistema cognitivo de visión para la navegación robótica", hemos extraído la explicación teórica del algoritmo SURF, así como su origen y mejoras significativas, mostradas a continuación en este capítulo. [17]

3.1 SURF (Speed Up Robust Feature)

3.1.1 Historia

SURF es un detector de variables locales y apareció con Herbet Bay en el 2006. Se inspira en el descriptor SIFT, pero presentan ciertas mejoras de su antecesor, como:

- Una velocidad en el cálculo notablemente superior, que no ocasiona perdida del rendimiento.
- Mayor resistencia ante posibles transformaciones de la imagen.

Estas mejoras se consiguen mediante la reducción de la dimensión y complejidad en el cálculo de los vectores de características de los puntos de interés obtenidos. Se reduce la información de trabajo pero continúan siendo igualmente característicos y repetitivos.

3.1.2 Algoritmo

En este apartado se pretende describir el procedimiento que sigue el algoritmo SURF para detectar los puntos de interés (keypoints), asignación de la orientación y por ultimo obtención del descriptor SURF.

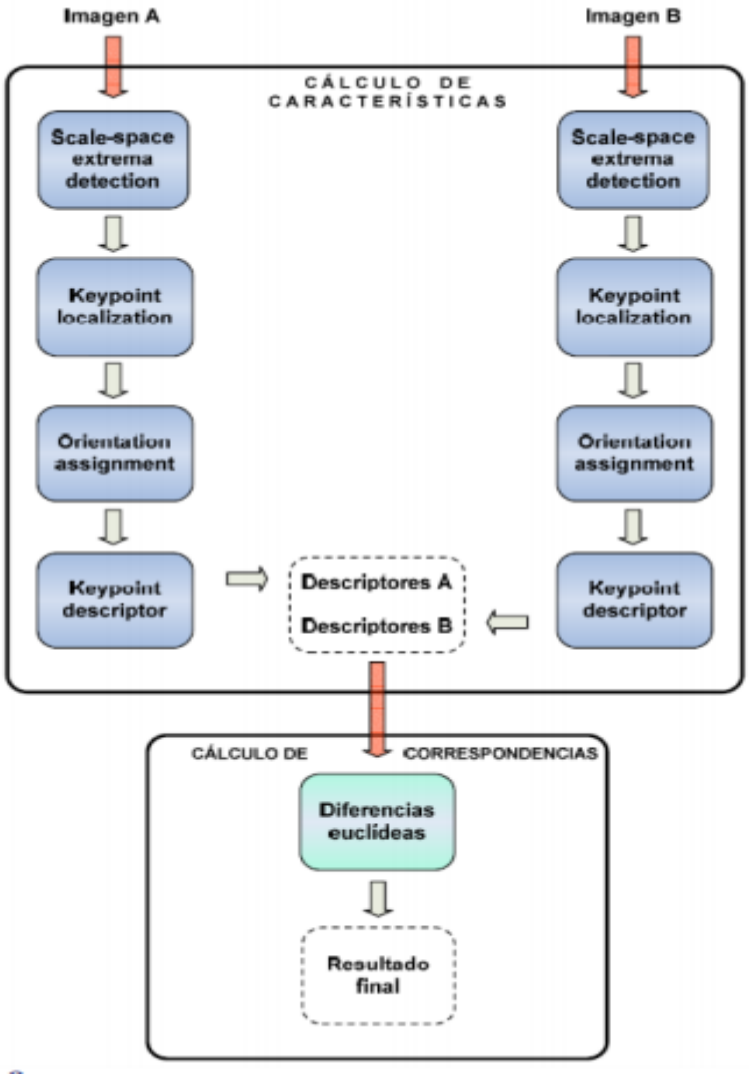


Figura 3-1 Esquema del proceso para realizar nuestro proyecto de clasificación SURF [17]

3.1.2.1 Detección de Puntos de Interés

La primera de las fases del descriptor SURF funciona igual que la del descriptor SIFT en cuanto a la detección de puntos de interés se refiere. El descriptor SURF hace uso de la matriz Hessiana, más concretamente, del valor del determinante de la matriz, para la localización y la escala de los puntos. El motivo por el que se emplea la matriz Hessiana será por su rendimiento en cuanto a la velocidad de cálculo y en cuanto a su precisión.

La principal novedad que presenta el descriptor SURF frente a sus antecesores es la forma de expresar los cálculos de la posición y la escala de los puntos característicos. SURF no usará diferentes medidas sino que usará el valor del determinante de la matriz Hessiana en ambos casos. Por lo tanto dado un punto p = (x, y) de la imagen, la matriz Hessiana H (p, o) del punto p perteneciente a la escala o se define como:

$$H(p,\sigma) = \begin{bmatrix} L_{xx}(p,\sigma) & L_{xy}(p,\sigma) \\ L_{yx}(p,\sigma) & L_{yy}(p,\sigma) \end{bmatrix}$$

Donde $L_{xx}(p,\sigma)$ será la convolución de segundo orden de la Gaussiana, $\frac{\delta^2}{\delta x^2} g(\sigma)$ con la imagen I en el punto x, y similarmente para $L_{xy}(p,\sigma)$ y $L_{yy}(p,\sigma)$.

Las aproximaciones de las derivadas parciales se denotan como Dxx, Dxy y Dyy, y el determinante se calcula de la siguiente manera.

$$\det(H_{aprox.}) = D_{xx}D_{yy} - (0{,}9\, D_{xy})^2$$

Donde el valor de 0,9 está relacionado con la aproximación del filtro Gaussiano.

En la imagen que aparece a continuación se puede observar la representación de la derivada parcial de segundo orden de un filtro gaussiano discretizado y la aproximación de la Derivada implementada en el caso del descriptor SURF.

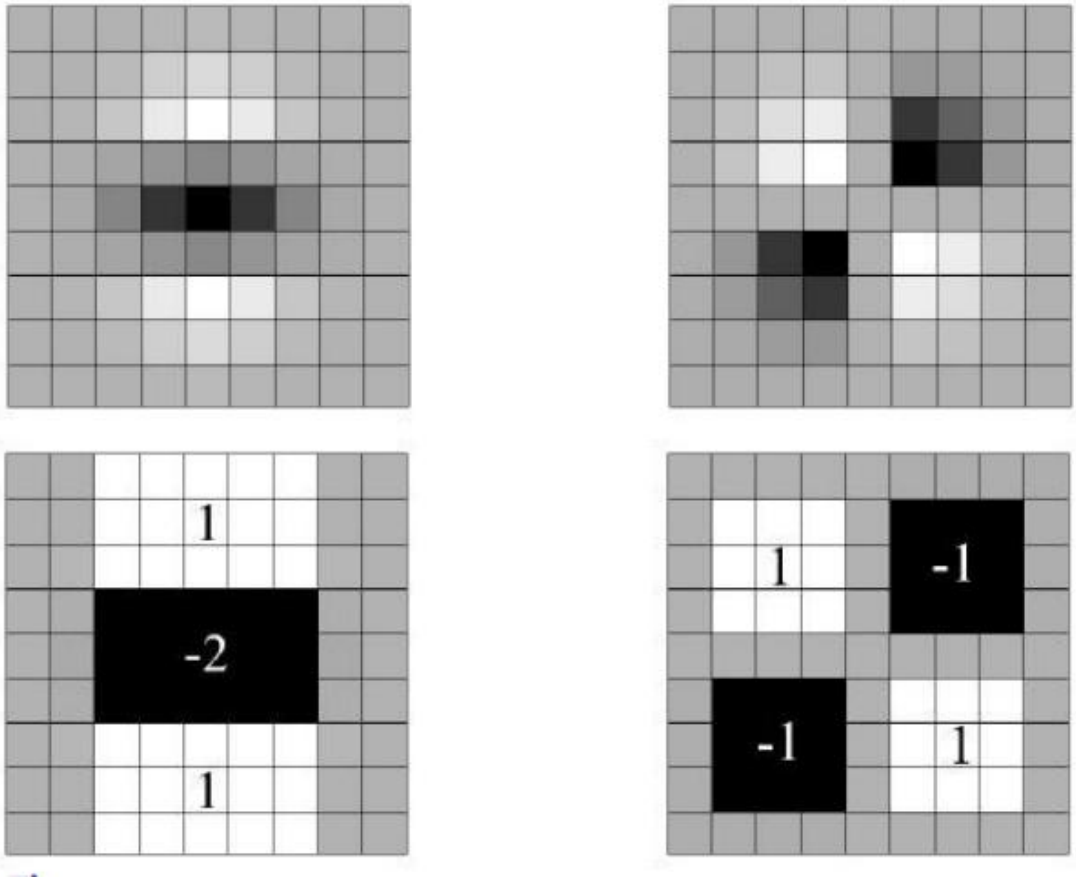


Figura 3-2 Representación de la derivada parcial de segundo orden de un filtro gaussiano. [17]

En la siguiente imagen se puede observar la representación de la derivada parcial de segundo orden de un filtro gaussiano discretizado y la aproximación de la derivada implementada en el caso del descriptor SURF.

La imagen de salida obtenida tras la convolución de la imagen original con un filtro de dimensiones 9 x 9, que corresponde a la derivada parcial de segundo orden de una gaussiana con o = 1:2, es considerada como la escala inicial o también como la máxima resolución espacial (s = 1:2, correspondiente a una gaussiana con o = 1:2).

Las capas sucesivas se obtienen mediante la aplicación gradual de filtros de mayores dimensiones, evitando así los efectos de aliasing en la imagen. El espacio escala para el descriptor SURF, al igual que en el caso del descriptor SIFT, está divido en octavas. Sin embargo, en el descriptor SURF, las octavas están compuestas por un número fijo de imágenes como resultado de la convolución de la misma imagen original con una serie de filtros cada vez más grande. El incremento o paso de los filtros dentro de una misma octava es el doble respecto del paso de la octava anterior, al mismo tiempo que el primero de los filtros de cada octava es el segundo de la octava predecesora.

Finalmente para calcular la localización de todos los puntos de interés en todas las escalas, se procede mediante la eliminación de los puntos que no cumplan la condición de máximo en un vecindario de 3 x 3 x 3. De esta manera, el máximo determinante de la matriz Hessiana es interpolado en la escala y posición de la imagen.

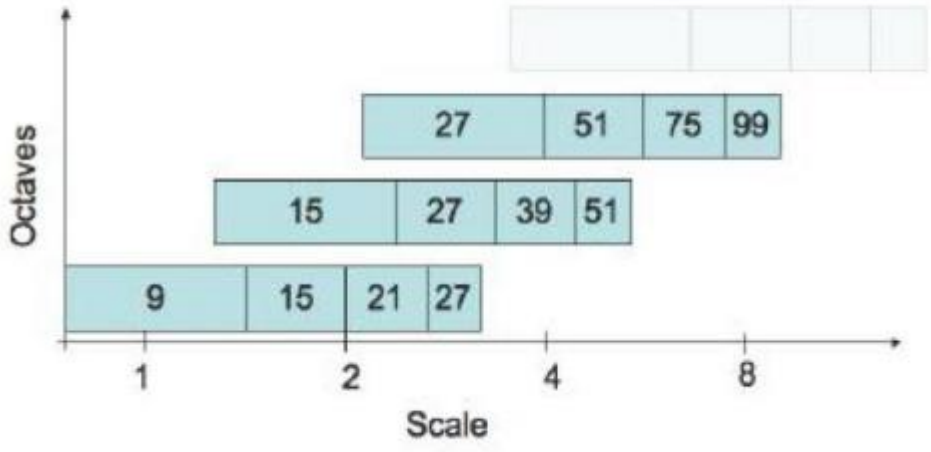


Figura 3-3 Escala para el descriptor SURF [17]

3.1.2.2 Asignación de la Orientación

La siguiente fase en la creación del descriptor viene dada por la asignación de la orientación de cada uno de los puntos de interés obtenidos en la fase anterior. Es en esta etapa donde se da al descriptor de cada punto la rotación invariante mediante la orientación del mismo.

El primer paso para otorgar la mencionada orientación consiste en el cálculo de la respuesta de Haar en ambas direcciones x e y mediante las funciones siguientes:

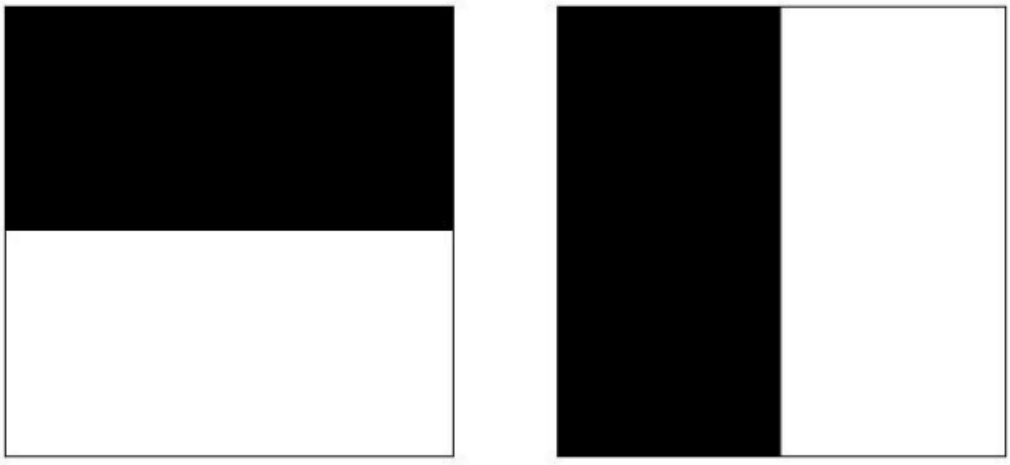


Figura 3-4 Cálculo de la respuesta Haar. Donde el color negro es el valor -1 y el blanco es +1. [17]

La etapa de muestreo depende de la escala y se toma como valor s. Se toma el valor 4s, siendo s la escala en la que el punto de interés ha sido detectado, por tanto dependiente también de la escala, como referencia, donde a mayor valor de escala mayor es la dimensión de las funciones onduladas.

Tras haber realizado todos estos cálculos, se utilizan imágenes integrales nuevamente para proceder al filtrado mediante las máscaras de Haar y obtener así las respuestas en ambas direcciones. Son necesarias únicamente 6 operaciones para obtener la respuesta en la dirección x e y. Una vez que las respuestas onduladas han sido calculadas, son ponderadas por una gaussiana de valor o = 2,5s centrada en el punto de interés.

Las respuestas son representadas como vectores en el espacio colocando la respuesta horizontal y vertical en el eje de abscisas y ordenadas respectivamente. Finalmente, se obtiene una orientación dominante por cada sector mediante la suma de todas las respuestas dentro de una ventana de orientación móvil cubriendo un ángulo de PI/3.

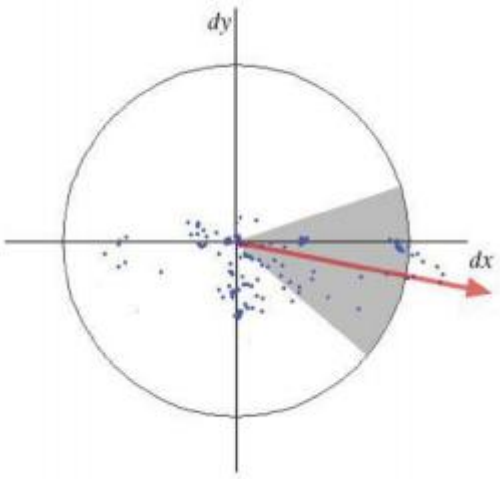


Figura 3-5 Cálculo direcciones x e y. [17]

3.1.2.3 Descriptores SURF

Se construye como primer paso una región cuadrada de tamaño 20s alrededor del punto de interés y orientada en relación a la orientación calculada en la etapa anterior. Esta región es a su vez dividida en 4 x 4 sub-regiones dentro de cada una de las cuales se calculan las respuestas de Haar de puntos con una separación de muestreo de 5 x 5 en ambas direcciones.

Por simplicidad, se consideran dx y dy las respuestas de Haar en las direcciones horizontal y vertical respectivamente relativas a la orientación del punto de interés.

Para dotar a las respuestas dx y dy de una mayor robustez ante deformaciones geométricas y errores de posición, éstas son ponderadas por una gaussiana de valor o = 3.3s centrada en el punto de interés. En cada una de las sub-regiones se suman las respuestas dx y dy obteniendo así un valor de dx y dy representativo por cada una de las sub-regiones. Al mismo tiempo se realiza la suma de los valores absolutos de las respuestas j dx j y j dy j en cada una de las sub-regiones, obteniendo de esta manera, información de la polaridad sobre los cambios de intensidad.

En resumen, cada una de las sub-regiones queda representada por un vector v de componentes:

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

Por lo tanto, englobando las 4 x 4 sub-regiones, resulta un descriptor SURF con una longitud de 64 valores para cada uno de los puntos de interés identificados.

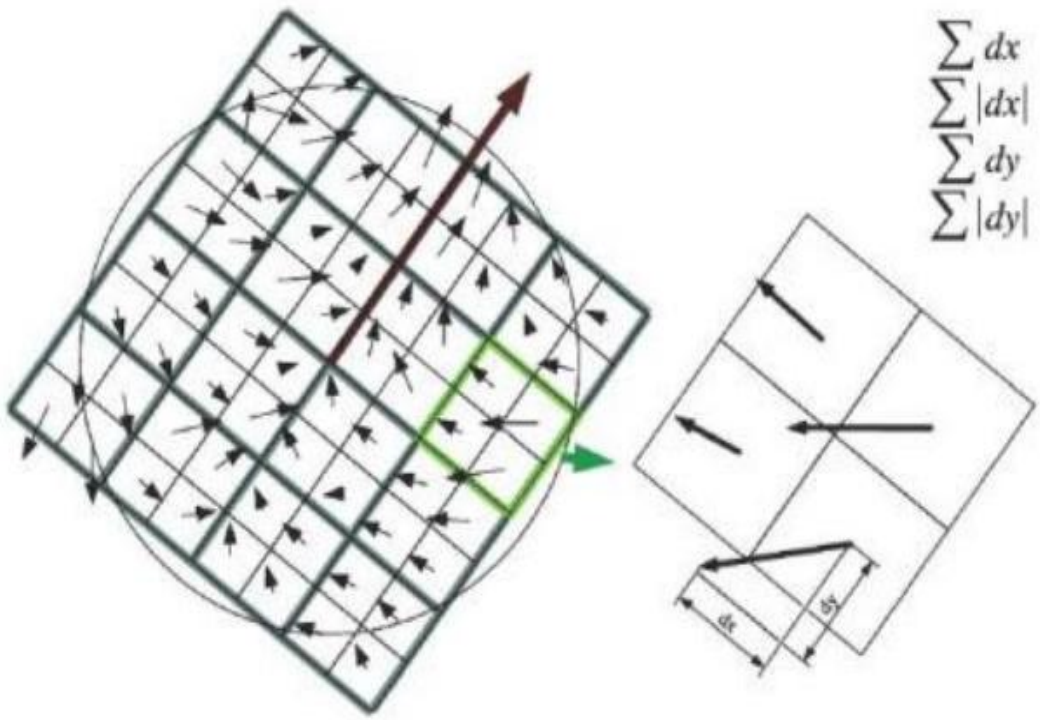


Figura 3-6 Descriptores SURF. [17]

3.2 RANSAC (RANdom SAmple Consensus)

RANSAC es un algoritmo iterativo que se utiliza para estimar los parámetros de un modelo matemático, de un determinado conjunto de datos, que contiene valores atípicos (*outliers*). Es usado a menudo en aplicaciones relacionadas con la visión artificial, por ejemplo, para resolver simultáneamente el problema de correspondencia y estimar la matriz fundamental.

Otros métodos para estimar parámetros de modelos como mínimos cuadrados le dan el mismo peso a todos los datos, por lo tanto la presencia de un *outlier* puede llegar a distorsionar el modelo obtenido. RANSAC es un algoritmo que se utilizó para estimar una homografía a partir de los puntos correspondientes obtenidos mediante algoritmos para detección de características.

Una homografía es una transformación proyectiva que determina una correspondencia entre puntos y así calcula la matriz de homografía. La matriz de homografía relaciona las coordenadas de un patrón con su respectiva imagen, y puede estimarse para cada imagen de un modelo 2D.

Suponemos que se quiere hallar una transformación afín unidimensional entre un conjunto de puntos posicionados entre dos líneas. El objetivo entonces será hallar una recta que minimice la suma de las distancias ortogonales al cuadrado, sujeto a la condición de que no haya ningún punto que se desvíe de la recta buscada más allá de una cierta distancia umbral. RANSAC, debido a ser un estimador robusto, es el que se usa en este programa para elegir los puntos homólogos ya que podrá usarse para casos en los que la proporción de puntos fuera del umbral sea muy grande.

Los algoritmos de detección de características tienden a sufrir dos tipos de errores, errores de clasificación y errores de medida.

- Los errores de **clasificación** se dan cuando el algoritmo de detección no identifica de forma correcta una zona de la imagen.
- Los errores de **medida** ocurren cuando el algoritmo de detección identifica correctamente la característica pero calcula erróneamente la ubicación en la imagen.

Los errores de medida se considera que tienen una distribución normal y podemos eliminarlos al trabajar con gran cantidad de puntos sin embargo, los errores de clasificación no cumplen ninguna distribución y provocan un efecto no deseado en el cálculo de la homografía.

3.2.1 Historia

El algoritmo RANSAC (RANdom SAmple Consensus) fue introducido por Martin L. Fischler y Robert C. Bolles en 1981 siendo éste un algoritmo iterativo para calcular los parámetros de un modelo matemático de un conjunto de datos observados que contiene valores atípicos. Es un algoritmo no determinista en el sentido de que produce un resultado razonable sólo con una cierta probabilidad, mayor a medida que se permiten más iteraciones.

La idea detrás del algoritmo es usar el menor número de puntos posibles para estimar el modelo y luego observar cuantos datos se ajustan correctamente al modelo estimado. En el caso de estimación de una homografía H se sortean cuatro puntos dentro de todas las correspondencias posibles y se calcula una homografía aproximada que llamaremos H'. Con esta homografía se aprecia la cantidad de puntos que, dentro de todas las correspondencias, son compatibles. Si la cantidad de puntos compatibles es suficiente, se utilizan estos puntos para estimar mejor la homografía H'.

3.2.2 Algoritmo

La entrada al algoritmo RANSAC está formada por un conjunto de valores de los datos que se obtienen tras las observaciones, y algunos parámetros de confianza[3]. RANSAC logra su objetivo mediante la repetición de los siguientes pasos:

1. Seleccionar un subconjunto aleatorio de los datos originales. Subconjunto llamado "inliers hipotéticos", o lo que es lo mismo, un conjunto de parámetros de confianza.
2. Se monta un modelo en dicho subconjunto.
3. Todos los datos que tenemos se prueban frente al modelo estimado. Los puntos que se ajusten al modelo se consideran como parte del conjunto de consenso.
4. El modelo estimado es bueno siempre que se hayan clasificado suficientes puntos como parte del conjunto de consenso.
5. Después, el modelo puede ser mejorado, volviendo a realizar una estimación, usando todos los miembros del conjunto de consenso.

Este procedimiento se repite un número determinado de veces, pudiendo producir o bien un modelo que es rechazado porque no hay suficientes puntos en el conjunto de consenso, o un modelo aceptado con suficientes puntos en el conjunto de consenso. En este último caso, mantenemos el modelo si su conjunto de consenso es más grande que el modelo guardado previamente.

[3] Llamamos *parámetros de confianza* al conjunto de valores dentro de un intervalo con alta probabilidad de éxito.

Como conclusión, del algoritmo RANSAC [18] tenemos tres parámetros para estimar:

- **Distancia entre el modelo ajustado y los puntos del subconjunto**: para considerar si un punto es compatible o no con dicho modelo. Se quiere una distancia tal que la probabilidad de que un punto sea atípico o distinto al deseado sea menor a α. En la práctica se asume que el error es gaussiano de media cero y varianza σ. Para un α de 0.95 el umbral utilizado para calcular la homografía será:

$$t = \sqrt{5{,}99}\sigma$$

- **Número de iteraciones**: para buscar subconjuntos de decisión. Se calcula en función de la probabilidad *p*, siendo *p* la probabilidad de que por lo menos un subconjunto de los que se elijan aleatoriamente, esté libre de *outliers* (valores atípicos o no deseados).

  ω es la probabilidad de que un punto del conjunto sea correcto, por lo tanto $\lambda = 1 - \omega$ es la probabilidad de encontrar un punto *outliers* (atípico).

  Para que se cumpla la probabilidad P se tiene que cumplir:

$$(1 - \omega^{s})^{N} = 1 - p$$

  Siendo N el número de iteraciones estimado y S el conjunto a analizar.

  Por lo tanto:

$$N = \frac{\log(1 - p)}{\log(1 - (1 - \lambda)^{S})}$$

- El **umbral, U**: para marcar el número mínimo, de puntos compatibles, para considerar una buena estimación del modelo. Dicho valor se busca que sea similar al número total de éxitos que existan en ese conjunto de datos. Por tanto, podemos calcularlo de la siguiente manera:

$$U = (1 - \lambda) * n$$

  donde n es la cantidad de puntos del conjunto de datos.

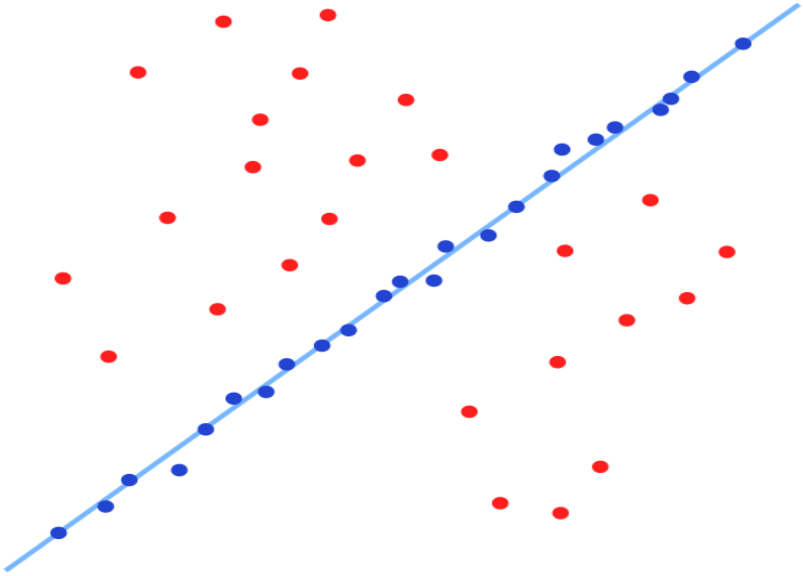


Figura 3-7 RANSAC. Puntos fuera (negro) y dentro (naranja) del Umbral. [21]

4 DETALLES DE IMPLEMENTACIÓN

Para algunos la vida es galopar un camino
empedrado de horas, minutos y segundos.
Yo, que más humilde soy, solo pido que la ola
que surge del último suspiro de un segundo
me transporte mecido hasta el siguiente….
-Roberto Iniesta-

Esta solución tiene como objetivo el diseño y la implementación de un sistema de visión artificial que permita posicionar en una escena y de forma autónoma una serie de patrones, almacenados previamente en una Base de Datos, obteniendo información acerca de su traslación, rotación, escala y coordenadas de profundidad. Dicha implementación se realiza a tiempo real, en la escena capturada por una cámara.

4.1 Introducción

En este programa se hace uso de la librería de visión artificial y código abierto OpenCV (The Open Computer Vision Library). La librería OpenCV proporciona un marco de trabajo de alto nivel para el desarrollo de aplicaciones de visión artificial en tiempo real, permitiendo facilitar el aprendizaje e implementación de las distintas técnicas. OpenCV proporciona bibliotecas de tipos de datos, tanto estáticos como dinámicos (matrices, grafos, árboles, etc.). Tiene compatibilidad para trabajar con la mayoría de cámaras del mercado y consigue entornos intuitivos en los dos SSOO (Sistemas Operativos) más populares del mercado: Microsoft Windows® y Linux.

El programa en si consta de tres etapas: reconocimiento de patrones de una Base de Datos; búsqueda y clasificación de un patrón y cálculos de contorno y posicionamiento dentro de la escena. En las siguientes figuras se muestra un par de esquemas donde aparecen las partes más relevantes de cada etapa:

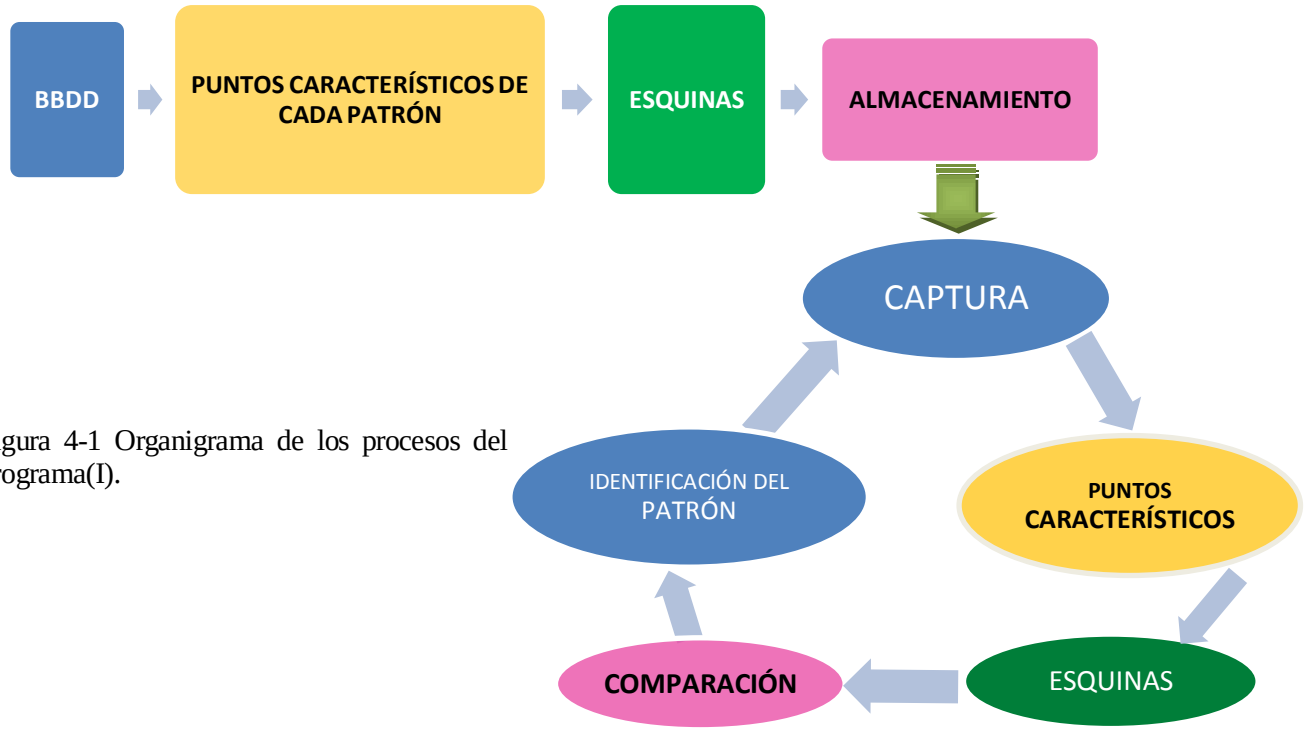


Figura 4-1 Organigrama de los procesos del programa(I).

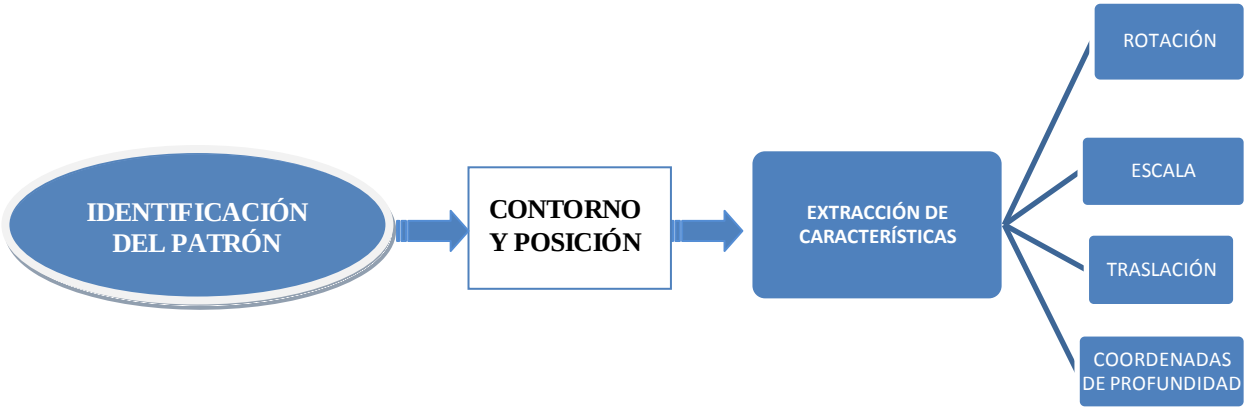


Figura 4-2 Organigrama de los procesos del programa(II)

Explicamos a continuación cada una de ellas así como las funciones principales que hemos creado para su implementación.

4.2 Reconocimiento de patrones en una base de datos

Se plantea un modelo de trabajo con la creación de una base de datos que incluye un patrón de cada elemento a identificar en la escena. Dicha base de datos podrá ser modificada, ampliada o reducida en función de las necesidades que se requieran en cada proceso de ejecución, sin necesidad de realizar en el código modificaciones. Para ello, usamos la función *getImagesFromDirectory* que, entregándole la dirección de la base de datos creada, nos devuelve un vector formado por los nombres de cada uno de los elementos del fichero.

El programa comienza con la captura de video a través de una webcam sincronizada a la aplicación y la llamada a la función *Inicializar*, la cual recibe de entrada, la dirección de la base de datos creada con anterioridad y cuatro variables pasadas por referencia que llamaremos: *V_descriptores*, *V_esquinas*, *V_keypoints_obj* y *NombresImg*. Esta función será la encargada de llamar a la función *getImagesFromDirectory* y las variables de referencia recibirán los valores correspondientes a:

- *V_descriptores*: vector de matrices, en el que se almacena la matriz con los descriptores de cada patrón de la BBDD.
- *V_esquinas*: Vector de vectores, el cual contiene en cada elemento el vector de las cuatro esquinas de cada patrón de la BBDD.
- *V_keypoints_obj*: Vector de vectores, que contiene en cada elemento el conjunto de los puntos característicos de cada patrón de la BBDD. Serán los valores de los puntos seleccionados y no la cantidad extraída.
- *NombresImg*: Vector de cadenas, que tendrá en cada elemento el nombre completo de cada patrón almacenado en la BBDD.

El funcionamiento interno de la función *Inicializar* es explicado posteriormente en el último apartado de este capítulo.

Aparecerá acto seguido un menú indicándonos el modo de inicializar la búsqueda de patrones y el modo de interrumpir la ejecución del programa.

4.3 Búsqueda y clasificación de un patrón

Se inicia la captura de imágenes en la que se irá haciendo, con cada una de esas capturas, el mismo procedimiento que utiliza la función *Inicializar* para la obtención de la matriz de nombre *descriptor_1* (donde almacenar el descriptor de la captura pertinente) y el vector *keypoints_scene* (para almacenar sus puntos característicos).

Tras el cálculo de dichos datos, el siguiente paso es la declaración del vector de vectores *matches* que se consigue con *BFMatcher*, un algoritmo de fuerza bruta disponible en la librería OpenCV. *BFMatcher* necesita para su funcionamiento una norma, que se utiliza para definir la distancia entre cada par de descriptores. En este programa hemos optado por usar NORM_L2, definida como distancia euclídea.

Para el descriptor de cada captura *BFMatcher* encuentra el descriptor más cercano entre los patrones de la Base de Datos, comparando cada uno de los descriptores de los patrones almacenados en ésta con el descriptor de la imagen capturada en cada instante. De esta forma, cada descriptor del primer set, queda emparejado con un único descriptor del segundo.

Como contamos con más de un patrón, el vector *matches* contendrá el resultado obtenido para cada intervención de *BFMatcher*. En las figuras 4-3 y 4-4 aparecen dos ejemplos del resultado que se obtiene tras seleccionar el patrón de más coincidencias con el objeto en la escena localizado:

Figura 4-3 Ejemplo 1 de una intervención de BFM

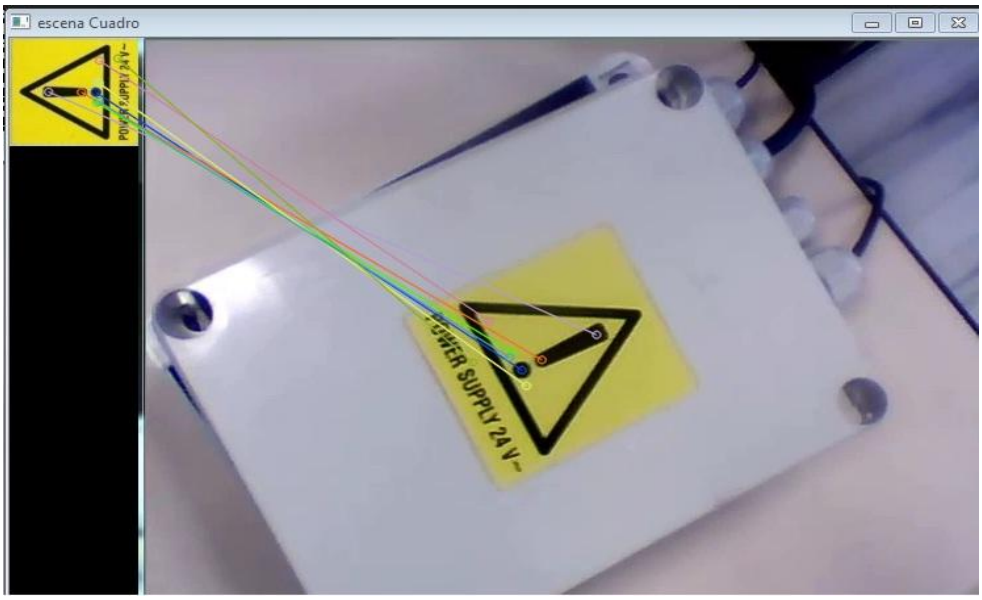


Figura 4-4 Ejemplo 2 de una intervención de BFMatcher.

El vector descriptor que obtenemos para los puntos de interés de cada patrón almacenado en la base de datos será SURF, ya que este algoritmo de visión artificial es el que presenta unos resultados más eficientes. Para la implementación del algoritmo SURF se ha escogido el lenguaje y entorno C++ por varias razones como son: velocidad, portabilidad (multiplataforma), ser compatible con las librerías de procesamiento de imagen como OpenCV, etc.

SURF extrae componentes descriptor para un conjunto dado de puntos de interés detectados. Recibe como entradas una representación integral de la imagen y un vector con los puntos de interés. Su salida consistirá en un vector con los descriptores SURF y los puntos de interés. Se implementa con los términos de OpenCV '*SurfFeatureDetector'* y '*SurfDescriptorExtractor* '. El primero se encarga de los procesos de detección de los puntos de interés, y el segundo término calcula el vector de características correspondientes a los puntos clave. *BFMatcher* será el encargado de hacer coincidir los vectores de características y *drawMatches* dibujará las coincidencias detectadas cuando un patrón haya sido reconocido dentro de la escena de captura.

4.3.1 Búsqueda de la máxima similitud

Contamos con los puntos característicos de cada patrón y los puntos característicos de la captura en cada instante. Se filtran los resultados validando solamente aquellos puntos cuya distancia de su correspondiente pareja no sea un valor fuera de la cota marcada por *(min_dist,max_dist)*. Se almacena cada resultado de tamaño numV[i] en un vector de vectores *good_matches* procediendo a la búsqueda de máxima similitud con los patrones.

Se compara la cantidad de puntos validados en cada patrón, para la escena capturada en ese momento. En esa comparación nos quedamos con los dos patrones más similares al objeto en cuestión. Será con el número de coincidencias de cada uno de estos dos patrones (número de puntos almacenados con los requisitos pertinentes) con los que comprobamos si la diferencia entre ambos es suficientemente elevada como para considerar al patrón de máxima similitud, el correspondiente al objeto localizado. Para ello se realizara un algoritmo de optimización.

4.3.2 Algoritmos de optimización

Para asegurar el correcto funcionamiento del programa así como su fiabilidad ante los resultados que nos proporciona, se desarrollan una serie de algoritmos que explicamos brevemente a continuación.

Quedándonos con el número de similitudes que tienen los dos patrones más afines al objeto en escena (*numMejor1* y *numMejor2*), consideramos sólo al patrón con *numMejor1* un resultado válido si su diferencia con *numMejor2* es lo suficientemente elevada para no dar opción a duda. Consideraremos suficiente esta diferencia si se cumple que:

La diferencia en valor absoluto entre ambos valores sea superior a un tercio de la suma de estos, es decir: si *dif* > *numMejor* , el patrón correspondiente a *numMejor1* será el correspondiente al objeto localizado en escena.

Siendo:
- Diferencia: *dif =abs(numMejor1 – numMejor2)*
- Un tercio de la suma de ambos valores: *numMejor= abs((numMejor1+numMejor2)/3)*

Este caso contempla hipótesis como la que se daría si ambos valores *numMejor1* y *numMejor2* fuesen iguales (Misma similitud encontrada con la captura). La diferencia entre ambas sería nula y por tanto nunca sería superior al tercio de su suma. Se descartaría el resultado e iniciaríamos el proceso de extracción de puntos característicos de la escena de nuevo.

Este filtro se aplica a los resultados que se obtienen de las herramientas anteriores como son *SURF* y *BFMatcher*, y reduce notablemente el índice de error sin suponer un retraso perceptible en la ejecución a tiempo real del programa. Una vez verificado el resultado, aparecen en pantalla los pares de puntos característicos entre ambas imágenes por medio de la función de OpenCV *drawMatches*, que dibuja las coincidencias encontradas de puntos clave a partir de dos imágenes, cuya llamada recibe de entrada:

```
drawMatches ( object, V_keypoints_obj[(iB-1)], pVideoFrame, keypoints_scene,
        good_matches [(iB-1)], img_matches, Scalar::all (-1), Scalar::all (-1),
        vector<char> (),DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS);
```

Siendo,

- *[(iB-1)]:* posición del patrón dentro de la Base de Datos.
- *object*: imagen del patrón seleccionado.
- *V_keypoints_obj[(iB-1)]:* puntos característicos del patrón seleccionado.
- *pVideoFrame:* imagen capturada por la cámara.
- *keypoints_scene:* puntos característicos de la imagen capturada por la cámara.
- *good_matches[(iB-1)]*: conjunto de pares de puntos entre las dos imágenes
- *img_matches* : imagen de salida. Su forma dependerá de los valores de formato que se den tras ella.
- Parámetros de formato: permiten elegir entre colores, formas de unión entre los puntos característicos,… (MatchColor, singlePointColor, matchesMask, Flags)

4.4 Cálculo de contorno y posicionamiento dentro de la escena

Llegados a este punto del proceso, ya hemos logrado identificar en la escena de captura el patrón de máxima similitud. Ya sabemos los puntos característicos coincidentes en ambas imágenes y eso, nos servirá de ayuda para el cálculo del contorno del objeto e información respecto a su escala y posicionamiento (traslación, rotación, coordenadas de 3D, salvo un factor de escala).

4.4.1 Cálculo del contorno

Para esta parte del proceso debemos primeramente detectar las esquinas del objeto localizado dentro de la escena. Ya contamos con las esquinas de las plantillas de la base de datos, calculadas al inicio del programa con la llamada a la función *Inicializar*, pero en este segundo caso la cosa se complica ya que el objeto no tiene por qué ocupar toda la escena capturada, ni situarse en la misma posición o distancia respecto al objetivo de la cámara. No debemos calcular todas las esquinas de la captura sino las del objeto de interés. Para ello, usamos el vector *scene* (goodpoints de la captura) y el vector *obj* (goodpoints del patrón seleccionado).

4.4.1.1 Detección de esquinas

Se realizará haciendo uso de la matriz de homografía, es decir, obteniendo una transformación proyectiva que determina una correspondencia entre las dos imágenes planas, de forma que a cada uno de los puntos de una de ellas le corresponderán un punto de la otra.

Obtendremos una imagen *H* a través de la función de OpenCV *findHomography* introduciendo en su llamada los puntos característicos de ambas imágenes: *scene* y *obj*.

```
Mat H = findHomography( obj, scene, CV_RANSAC, 3 );
```

El método que introducimos nosotros para esta aplicación es *CV_RANSAC*, un método robusto basado en RANSAC- y el error máximo de re-proyección , *ransacReprojThreshold*, tendrá un valor de rango del 1 al 10. Nosotros hemos definido *ransacReprojThreshold=3*, utilizado sólo con el método RANSAC. Este valor dependerá mucho de los objetivos y requisitos de la aplicación. Puede ser recomendable establecer un umbral muy alto si se tiene luego intención de refinar el resultado utilizando algún otro método más tarde. Si el resultado es el valor final, y se necesita una relativa precisión, entonces el umbral deberá ajustarse mucho más bajo.

Al finalizar este paso las dos imágenes se encuentran en un mismo espacio de referencia y podemos obtener las regiones de solapamiento entre ellas mediante la imagen *H*.

Necesitamos un vector en el que se almacenen las esquinas de nuestro objeto localizado, al que llamaremos *scene_corners*. Será pasado por referencia, en la llamada a la función *PerspectiveTransform*, la cual recibe la matriz de homografía *H* y las esquinas de la plantilla seleccionada *V_esquinas[(iB-1)]*.

Para trabajar con las esquinas del objeto se suma a los valores de *scene_corners* los desplazamientos dentro de la ventana del programa. Haremos uso de *V_esquinas[iB-1]*.

```
Point2f esq1 = scene_corners[0] + Point2f( object.cols, 0);
```

4.4.1.2 Formación de un contorno

El objetivo de esta parte consiste en calcular el contorno de los objetos localizados formando un polígono regular de cuatro lados. Teniendo las esquinas del objeto obtenidas en el apartado anterior, esto se conseguiría aplicando una función de OpenCV llamada *rectangle*.

El problema viene después, ya que las imágenes se capturan desde una cámara en continuo movimiento. La traslación y rotación del objeto son factores que deben tenerse en cuenta. Con la función *rectangle* el objeto quedaría fuera del contorno si su posición se gira o desplaza en la escena.

Como solución a este problema, el contorno rectangular se formará por cuatro líneas, cuyos extremos irán unidos dos a dos a cada vértice, mediante la función *line* se genera una línea entre los dos puntos indicados en su llamada.

Se usa una función llamada *pendiente* para obtener la inclinación de cada una de las cuatro líneas, llamándose *m1, m2, m3 y m4.* En el caso de que no coincidan las pendientes de cada par (*m1=m3;m2=m4*), el polígono resultante no sería un contorno válido por lo que se descartaría el resultado. Con este sencillo filtro garantizamos de nuevo el buen funcionamiento del programa.

En las figuras 4-5 y 4-6 se observan los resultados tras conseguir un contorno cuadrilátero y adaptativo a la orientación que presente el objeto en escena:

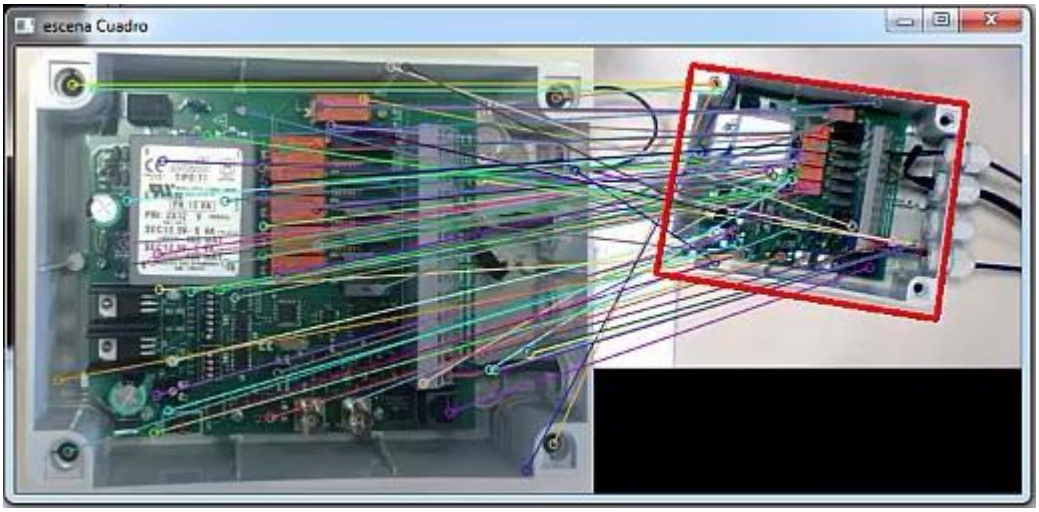


Figura 4-5 Ejemplo 1 de cálculo del contorno adaptativo a distintas orientaciones.

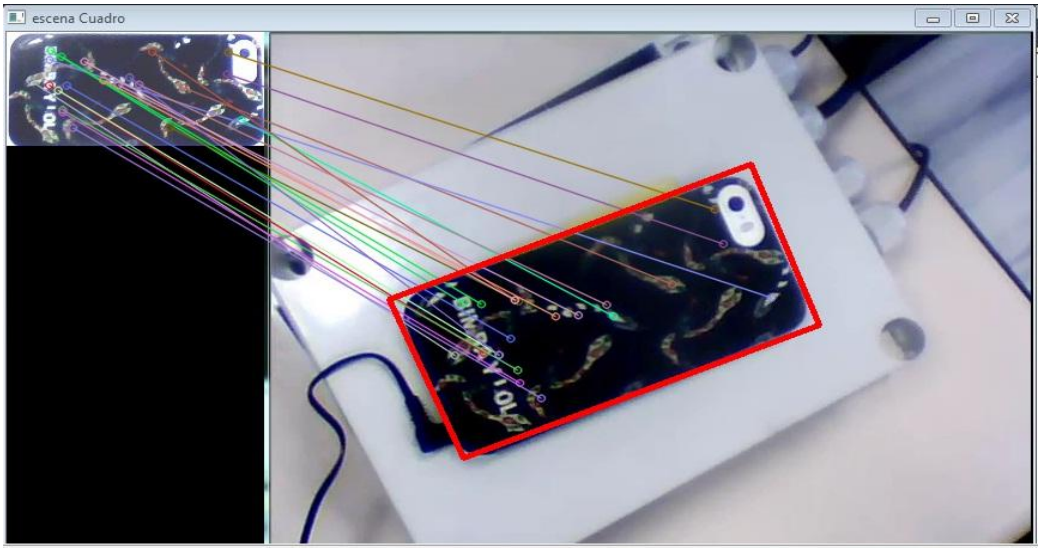


Figura 4-6 Ejemplo 2 de cálculo del contorno adaptativo a distintas orientaciones.

4.4.2 Obtención de las características de posicionamiento

La información que conseguimos obtener con este programa consiste principalmente en escala, posicionamiento y coordenadas 3D, salvo un factor de escala.

Para la escala y condiciones de posicionamiento (traslación y rotación) se llama a la función *ExtracTransH*, explicada con más detenimiento en el apartado 4.5.4 , la cual, recibe como parámetros de entrada la transformada de la matriz homográfica *TransH* y los parámetros por referencia que devuelven en dos vectores la información de escala, traslación y en una variable *double*, la rotación.

Para la obtención de la matriz de matrices de coordenadas de profundidad hemos usado una nueva función de OpenCV(Versión OpenCV3.0) llamada *decomposeHomographyMat*. La función *decomposeHomographyMat* extrae el movimiento de la cámara, relativo entre dos puntos de vista, observando primero un objeto plano (de la homografía *H*). Ésta función también es capaz de calcular la rotación y traslación del objeto, pero resulta muy útil por incorporar en ella un cálculo aproximado de las coordenadas 3D.

De entrada recibe la matriz *H* antes calculada y *K*, siendo *K* la matriz de calibración de la cámara de entrada intrínseca. Para el cálculo de *K* se deben estimar los parámetros intrínsecos y extrínsecos de la misma cámara, los cuales son necesarios para realizar la reconstrucción 3D del entorno y situar la cámara en la misma posición.

4.5 Funciones externas al programa principal

4.5.1 getImagesFromDirectory

Función externa al programa principal que es llamada por la función, también externa, de nombre *Inicializar*. Se le pasa como entrada una cadena de caracteres que forman una dirección hacia la Base de Datos con la que se quiere trabajar. Esta función devuelve un vector de cadenas en el que se almacenan los nombres de cada patrón en cada elemento del vector. Este vector lo recibe la función *Inicializar*, tras su llamada.

4.5.2 Inicializar

Función externa al programa principal que es llamada al inicio del programa por éste mismo, la cual, recibe la entrada de la dirección de la base de datos creada, y cuatro variables pasadas por referencia que llamaremos: *V_descriptores*, *V_esquinas*, *V_keypoints_obj* y *NombresImg*. Las variables de referencia recibirán los valores correspondientes a:

- *V_descriptores*: Vector de matrices, en el que se almacena la matriz con los descriptores de cada patrón de la BBDD.
- *V_esquinas*: Vector de vectores, el cual contiene en cada elemento el vector de las cuatro esquinas de cada patrón de la BBDD.
- *V_keypoints_obj*: Vector de vectores, que contiene en cada elemento el conjunto de los puntos característicos de cada patrón de la BBDD. Serán los valores de los puntos seleccionados y no la cantidad extraída.
- *NombresImg*: Vector de cadenas, que tendrá en cada elemento el nombre completo de cada patrón almacenado en la BBDD.

Al inicio de la función, se llama a la función *getImagesFromDirectory*, la cual devuelve un vector de cadenas correspondiente a la variable *NombresImg*, con el nombre completo de cada patrón almacenado en la Base de Datos como elementos del vector.

Después la función recorre en un bucle *for*, uno a uno los elementos del vector *NombresImg*, calculando el descriptor y los puntos característicos de cada imagen y almacenándolos en el vector *V_descriptores* y en el vector *V_keypoints_obj* correspondientemente.

Para ello hacemos uso del extractor y el detector del algoritmo SURF, mediante las siguientes expresiones:

```
SurfFeatureDetector detector( minHessian );
SurfDescriptorExtractor extractor;
```

Donde el valor del umbral *minHessian* lo hemos fijado en 900, tras una serie de pruebas para encontrar los resultados óptimos.

El bucle acaba con el cálculo de las esquinas de la respectiva imagen, crea un vector *img_corners* donde se introducen los cuatro puntos de las cuatro esquinas, cada uno de ellos con sus dos coordenadas, siendo de tipo Point2f.

```
img_corners[0] = cvPoint(0,0);
img_corners[1] = cvPoint( img.cols, 0 );
img_corners[2] = cvPoint( img.cols, img.rows );
img_corners[3] = cvPoint( 0, img.rows );
```

Y como en el caso anterior, este vector *img_corners* se irá guardando en cada iteración del bucle *for* en las posiciones del vector *V_esquinas* devolviendo estos resultados al programa principal tras haber recorrido todas las imágenes de la Base de Datos.

4.5.3 Pendiente

Función externa al programa principal que es llamada en el proceso de creación del contorno, la última fase, la cual recibe la entrada dos puntos como variables de tipo Point2f.

Calcula la pendiente de la recta que genera la unión de dichos puntos de forma aritmética, a través de sus vectores, y la devuelve al programa. Esta función se usa para optimizar el resultado final, dando posibilidad de realizar un contorno rectangular a base de 4 líneas, que cumplan el requisito, dos a dos, de tener pendiente común entre sí.

4.5.4 ExtracTransH

Esta función nos permite obtener la traslación, rotación y escala de un objeto en 2D, por lo que se debe complementar con una función de OpenCV *decomposeHomographyMat* que contemple también las coordenadas de posicionamieno del objeto de la imagen, proporcionándonos información para posibles reconstrucciones futuras.

En la función *ExtracTransH* aplicamos aritméticamente la forma de extracción de valores de rotación y escala a partir de una matriz de transformación 2D. Para ello se debe resolver lo siguiente:

$$\begin{bmatrix} a & b & tx \\ c & d & ty \end{bmatrix} = \begin{bmatrix} Sx * cos\varphi & -Sx * sin\varphi & Xc \\ Sy * sin\varphi & Sy * cos\varphi & Yc \end{bmatrix}$$

Donde S_x y S_y son las proporciones de escala, y x_c e y_c son la traslación y rotación del ángulo φ:

$$x_c = tx$$

$$y_c = ty$$

$$S_x = sign(a) * \sqrt{a^2 + b^2}$$

$$S_y = sign(d) * \sqrt{c^2 + d^2}$$

$$tan\varphi = -\frac{b}{a} = \frac{c}{d}$$

Así que el ángulo será tal que:

$$\varphi = atan2(-b, a) = atan2(c, d)$$

Y esos valores son los que devuelve la función tras su llamada, en funciones pasadas por referencia.

5 SIMULACIONES Y ANÁLISIS DE LOS RESULTADOS

Saltar rápidamente a conclusiones rara vez conduce a felices aterrizajes

-S. Sipurin-

El apartado de pruebas es vital en un trabajo de investigación, puesto que en él nos encargamos de comprobar que se cumplen todos los requisitos y exigencias antes de dar el proyecto por finalizado.

El proyecto se ha realizado utilizando una webcam, modelo XPRESSCAM 300 de la marca NGS con sensor CMOS de 300Kpx. El Sistema Operativo ha sido Windows 7 y el software Microsoft Visual Studio 2010. A continuación, se describirán los procedimientos que se han empleado para probar el sistema y garantizar así la calidad del resultado final en cada fase del programa.

5.1 Lenguaje de Programación

El lenguaje de programación elegido ha sido C++ frente a Python. La decisión se ha tomado debido a que buscábamos un lenguaje con gran potencial. Python se caracteriza por una mayor facilidad de uso y ahorro de tiempo a la hora de programar. Nos interesaba más capacidad computacional así que nos posicionamos por el uso de C++, a pesar de ser menos intuitivo.

La prioridad en el plan de pruebas será la comprobación de la totalidad del código programado, evitando así cualquier error de compilación, ahorrando situaciones no esperadas al ejecutar el programa. Para ello se han ido comprobando cada una de las fases por separado y su solapamiento con el resto se ha implementado en cada paso, dentro del desarrollo del programa.

5.2 Estructura del programa

En el programa podemos diferenciar dos partes: la ventana de diálogo y la ventana donde es proyectada la escena capturada por la cámara. La primera ventana establece diálogo con el usuario. Dicha ventana se muestra en la figura 5-1. Aparece al iniciar el programa, explica brevemente su funcionamiento y se queda a la espera de que se dé la orden para que inicie la actividad. La segunda es donde se proyecta la escena a analizar en el programa.

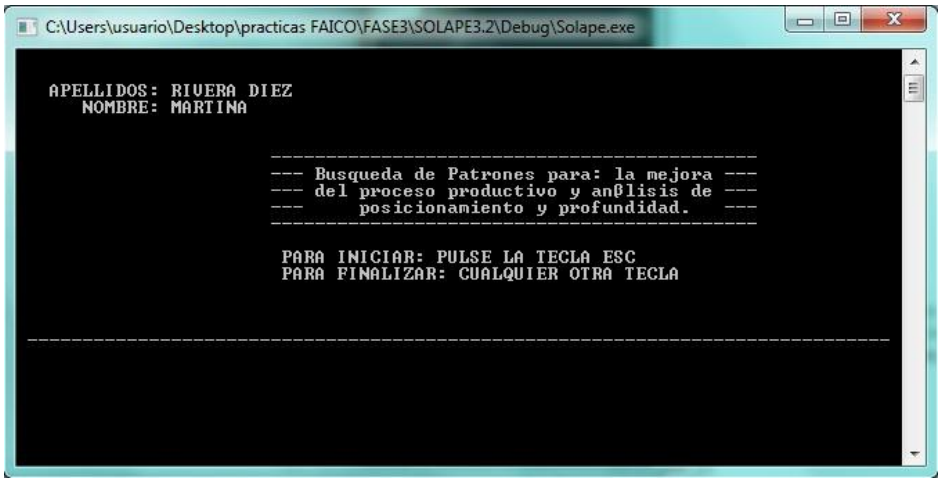


Figura 5-1: Menú de Inicio del Programa

Al inicializar el proceso, con la pulsación de la tecla *Escape*, se inicia el proceso de capturas simultáneas de la escena. Se realizará el emparejamiento con cada una de ellas siendo, después, comparadas con los patrones de la Base de Datos almacenados.

La ventana de diálogo puede mostrarnos, en el trascurso del proceso, el patrón que está siendo asignado al objeto de la escena así como la información de rotación, traslación, escalado y las coordenadas 3D de forma matricial, salvo un factor de escala.

En la imagen inferior 5-2 se muestra el proceso de reconocimiento del programa. En este caso se toma el criterio de seguir un determinado orden en la aparición de cada patrón, simulando un proceso de montaje. En caso de detectar un patrón fuera de su turno te indica la fase actual y la fase siguiente. En caso de haber detectado la fase deseada te indica cual es el siguiente paso. Aquí sería sencillo incluir con cada fase una descripción de la fase siguiente para un uso más sencillo e intuitivo del usuario.

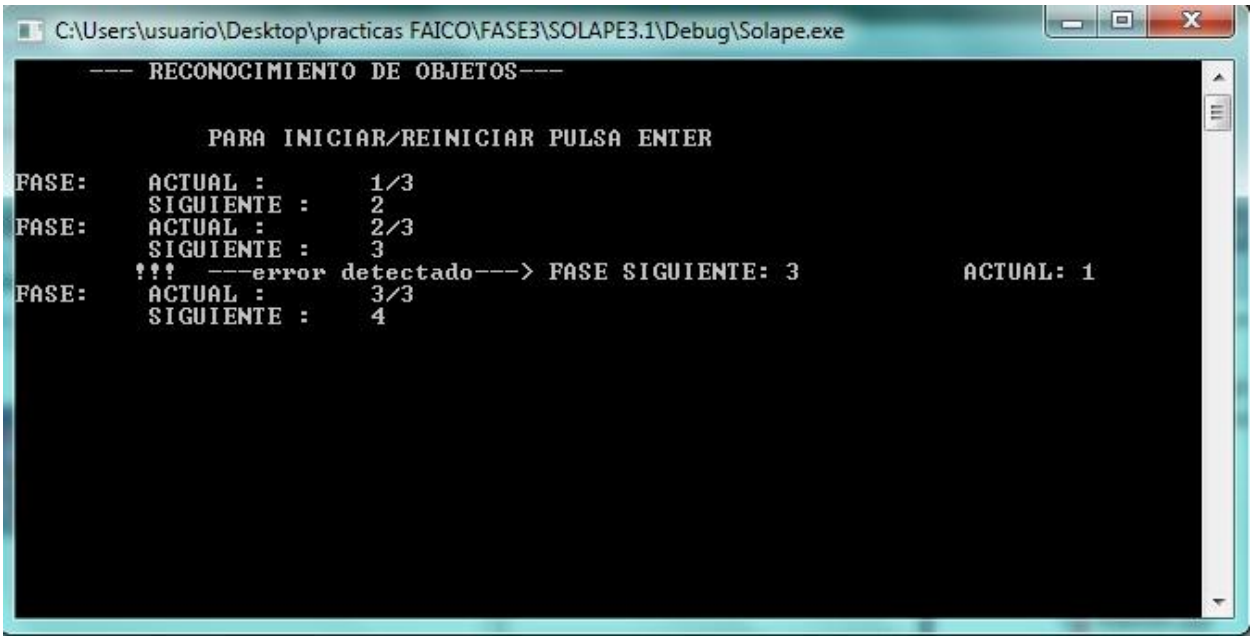


Figura 5-2: Detección de fases e instrucciones a seguir.

De cada fase o patrón detectado correctamente es extraída la información que nos interesa:

- En la imagen 5-3 aparece la matriz de componentes de profundidad 3D del objeto localizado dentro de la escena.
- En la imagen 5-4 aparece información sobre la escala, traslación y rotación del objeto localizado dentro de la escena.

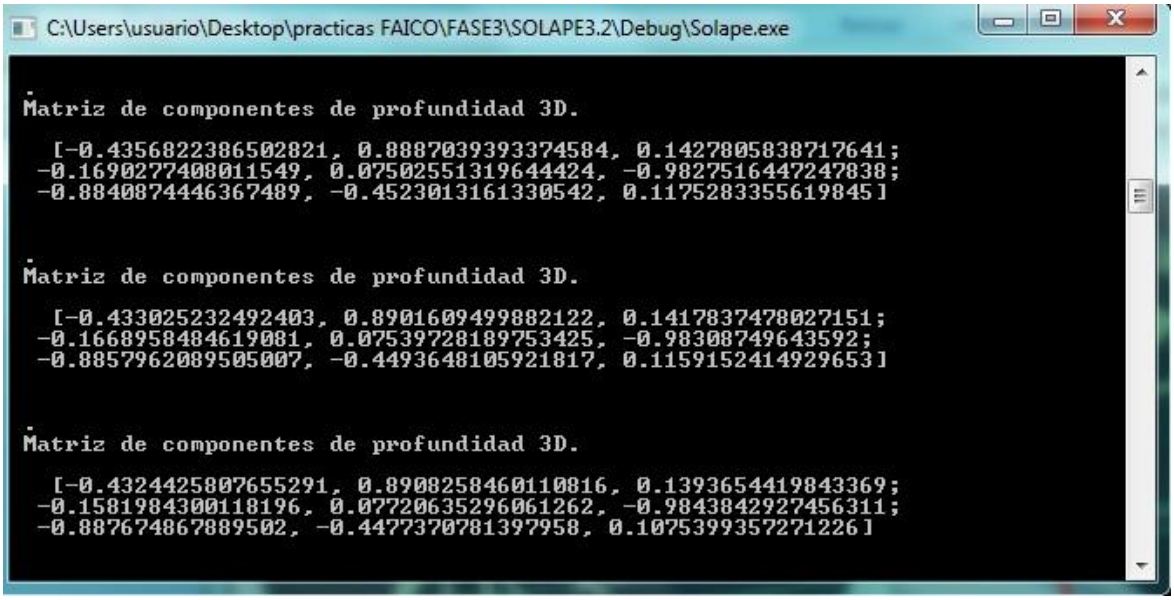


Figura 5-3: Matriz de componentes de profundida.

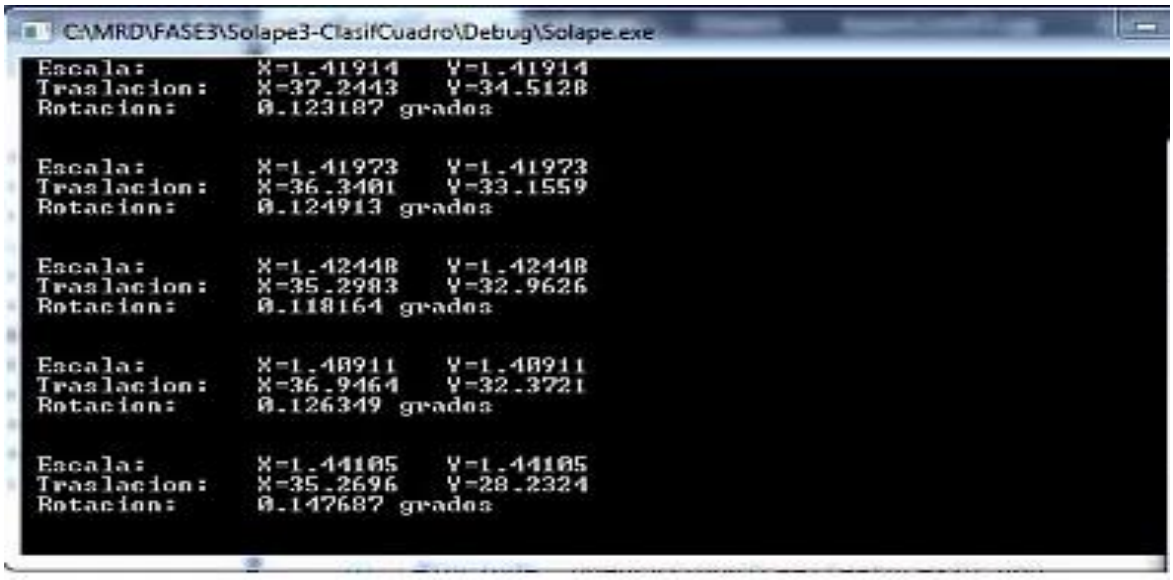


Figura 5-4: Información de la escala, traslación y rotación del objeto localizado dentro de la escena.

5.3 Pruebas de calidad y desempeño del sistema

Para las pruebas de calidad del sistema y su correcto desempeño hemos probado con distintos métodos y umbrales, evaluando cual obtenía los resultados óptimos. Se tienen en cuenta dos grupos de errores que nos interesa diferenciar:

1. **Falsos positivos**. Se trata de resultados que han sido devueltos incorrectamente, es decir, que han sido reconocidos como pertenecientes a una determinada clase o a un determinado punto de emparejamiento, pero no lo son. Vemos dos ejemplos en las figuras 5-5 y 5-6.

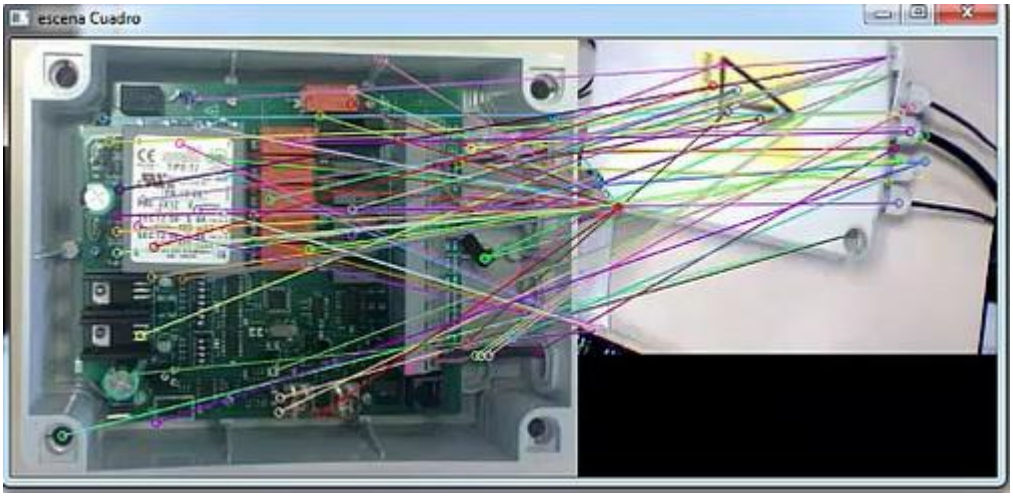
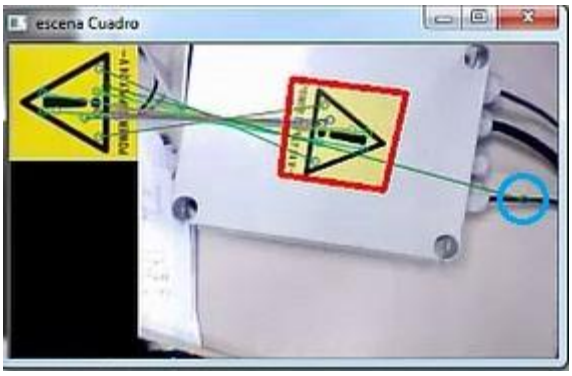


Figura 5-5: Falso positivo (Ejemplo A).

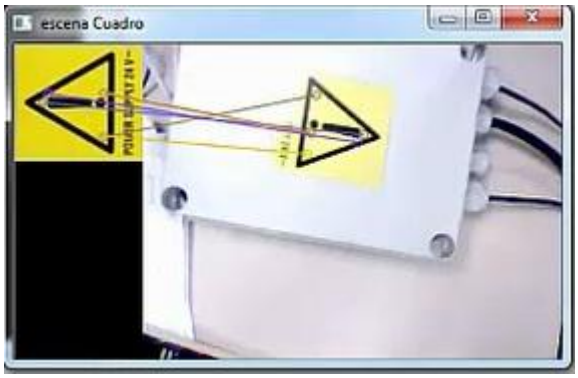


En el ejemplo A se muestra como el número de puntos característicos coincidentes, entre patrón y objeto localizado, puede en algunos casos no ser correcto y así designar erróneamente un modelo.

En el ejemplo B se muestra rodeado con un círculo azul un punto característico asignado a un punto de la escena al que no se corresponde.

Figura 5-6: Falso positivo (Ejemplo B).

2. **Falsos negativos**. Son imágenes que no son reconocidas como pertenecientes al tipo consultado, o lo son pero con un bajo nivel de aceptación, cuando en realidad deberían ser reconocidas sin problemas. En la imagen 5-7 vemos un claro ejemplo de este tipo.



En la imagen 5-7 observamos que, pese a ser identificado el patrón correcto en la escena, la escasez de puntos característicos encontrados entre ambas imágenes no nos permite obtener la información necesaria para calcular los resultados esperados.

Estos errores suelen deberse a la falta de nitidez de las capturas obtenidas de la cámara, o a valores de iluminación mayores o menores de los deseados.

Figura 5-7: Falso negativo

Los errores presentados a lo largo del proceso son corregidos. Se explican con detalle los métodos de optimización en el apartado 5.3.2.

5.3.1 SIFT vs. SURF

Para empezar se hicieron pruebas comparativas entre los métodos SIFT y SURF. Ambos son algoritmos de visión artificial que pertenecen a las librerías de OpenCV.

Pese a que los resultados con el método SIFT eran bastante buenos, si llevamos el problema a la comparación de centenares de imágenes, sus diferencias con el método SURF, en cuanto a velocidad de ejecución, se pueden considerar relevantes. Desde un punto de vista de obtención de puntos característicos detectados, el algoritmo SIFT sobrepasa a SURF, véase en las figuras 5-8 y 5-9, pudiendo detectar más del doble de puntos. Esto se produce porque SURF contempla la posibilidad de que haya varios puntos invariantes en una misma posición con diferente escala u orientación entre ellos.

La diferencia de puntos detectados hace que SIFT utilice mucho más tiempo para el análisis de una imagen que SURF. En un caso con 10 imágenes el tiempo de analizar una imagen con SIFT sería de 0.1646 ms mientras que en el caso de trabajar con SURF tendría una duración de casi un tercio que la anterior. A priori puede parecer una diferencia insignificante pero si ampliamos la Base de Datos a 100 o 1000 imágenes el retraso ocasionaría un empeoramiento notable en el rendimiento del programa. Este factor es muy importante para el objetivo del proyecto, ya que se implementa en tiempo real y sin restricciones en el tamaño de su Base de Datos.

Dentro del descriptor y detector utilizado, se necesita el umbral al que llamamos *minHessian* dentro de nuestro programa. En la decisión de si un pixel de una imagen es un punto significativo o no, la aproximación de la matriz de Hesse está construida con las derivadas parciales de las intensidades de la imagen, el determinante de esta matriz se llama Hesse y nos indica cómo de robusto es el pixel. El valor de *minHessian* sirve para decidir el valor de puntos clave que se obtendrán. A mayor valor, menor cantidad de puntos pero de mayor fiabilidad. A menor valor, mayor cantidad de puntos pero con mayor riesgo de presentar ruido. El valor más usado oscila entre los 400 y 1000. Tras pruebas de verificación fijamos el valor de 900 ya que los resultados iban mejorando progresivamente al incrementar la variable, hasta alcanzar este valor donde se llegó al resultado óptimo.

Durante el desarrollo del programa hemos trabajado con videos almacenados previamente en memoria y con capturas a tiempo real de una escena a través de una webcam.

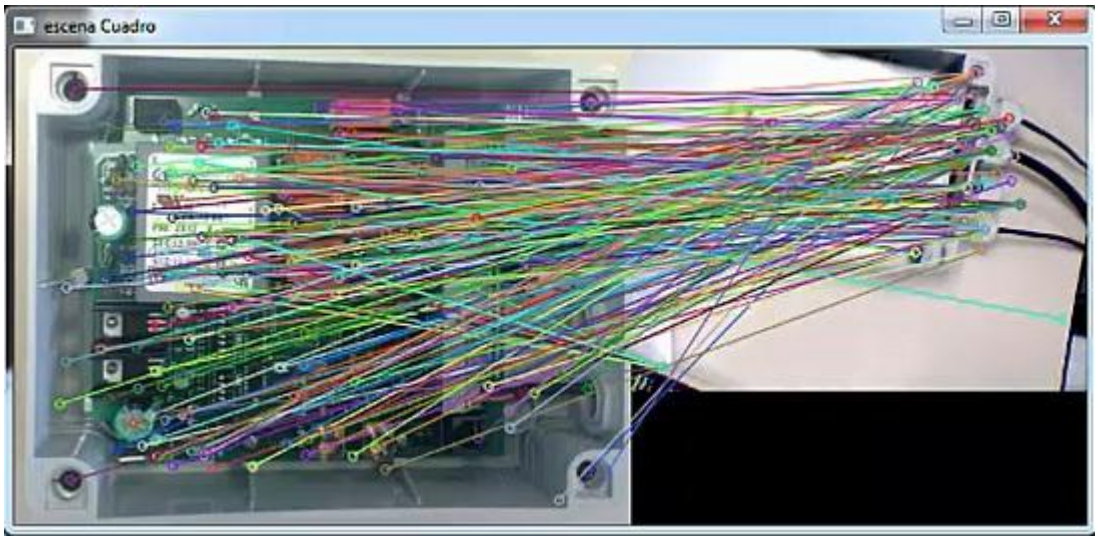


Figura 5-8: Obtención de puntos característicos detectados por el algoritmo SIFT.

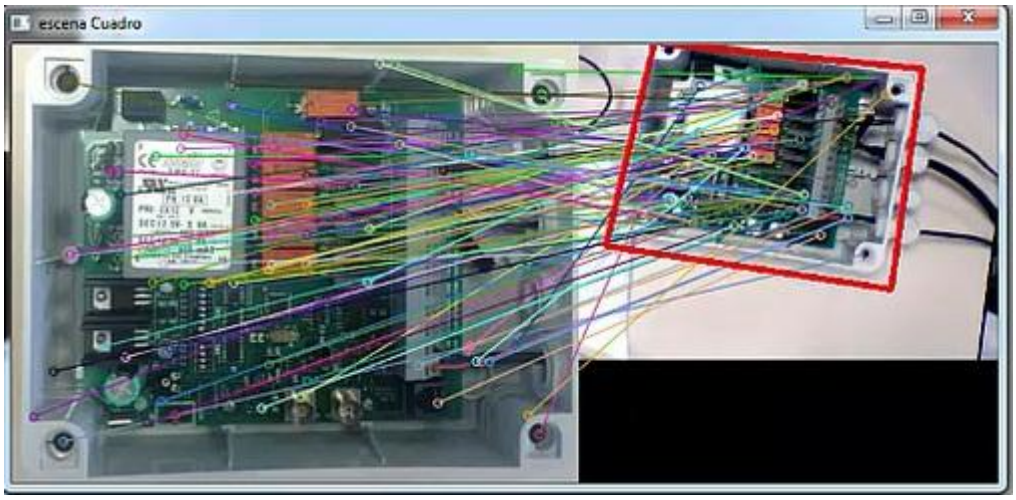


Figura 5-9: Obtención de puntos característicos detectados por el algoritmo SURF.

5.3.2 Algoritmos de Optimización

Tras poner en marcha el proceso de comparación entre las imágenes de la Base de Datos y la captura de la cámara, se procede a la identificación del patrón capturado en la escena. Ésta asignación se hará en función del número de puntos característicos emparejados entre dos imágenes, patrón y captura.

Se consigue un resultado con un porcentaje superior al 95% de aciertos tras aplicar los algoritmos de optimización explicados anteriormente en el apartado 4.3.2. Ante situaciones de ambigüedad entre dos patrones o ante una insuficiencia de pares de puntos entre captura y patrón, los resultados se descartan. La velocidad del programa permite la supresión de capturas no válidas sin tener un cambio apreciable a simple vista en los resultados. Se consideran capturas no válidas aquellas que, debido al ruido, ambigüedad o desenfoque de la escena, impiden un reconocimiento del patrón. También serán descartadas aquellas escenas donde no aparezca ningún objeto de interés, es decir, ningún objeto almacenado dentro de la Base de Datos.

Para evitar falsos positivos de tipo A se exige al programa que los resultados, en la identificación del mismo, sean constantes durante 5 ms. El error debe repetirse de forma reiterada para que influya en los resultados. De todas formas, para la extracción de información se debe cumplir un mínimo de pares de puntos comunes que, un patrón distinto al deseado, no ha sido capaz de alcanzar durante las simulaciones realizadas en el proceso.

Para los falsos positivos de tipo B se hizo un intervalo más restrictivo sobre las distancias permitidas entre los puntos característicos a comparar. Así, se vio reducido este tipo de errores consiguiendo que no tengan efecto en los resultados finales.

Para evitar los falsos negativos se cuidaron las condiciones del entorno de estudio, así como la calidad de imagen de los modelos almacenados en la Base de Datos.

5.3.3 Cálculos del contorno

Los cálculos del contorno de los objetos detectados se basan en el análisis de los puntos característicos extraídos de la captura de la cámara. Los puntos periféricos del conjunto serán la referencia para obtener el perímetro de su superficie.

Para simplificar el problema se calcula un rectángulo que bordee la figura. Dicho rectángulo tiene que ser sensible a posibles rotaciones del objeto durante la secuencia de imágenes. Para ello se necesitarán líneas creadas de forma independiente que puedan tener pendiente no nula, garantizando que sean paralelos entre sí, cada par de lados opuestos.

Para evitar contornos no regulares se descartan aquellos resultados cuyo resultado de las pendientes no resulten comunes entre cada par de lados opuestos. Con este planteamiento se consigue un contorno perfecto en el trascurso de la ejecución. En las figuras 5-10 y 5-11 se muestra la forma obtenida antes de aplicar el algoritmo de optimización de la función *pendiente* (4.5.3) en el cálculo del contorno. En la figura 5-12 se muestra la situación que aparecía cuando la formación del rectángulo se generaba con la función *rectangle*, de OpenCV.

Figura 5-10: Error de contorno. Ejemplo A.

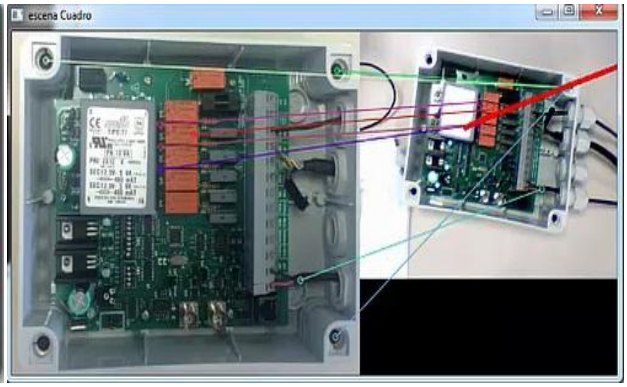


Figura 5-11: Error de contorno. Ejemplo B.

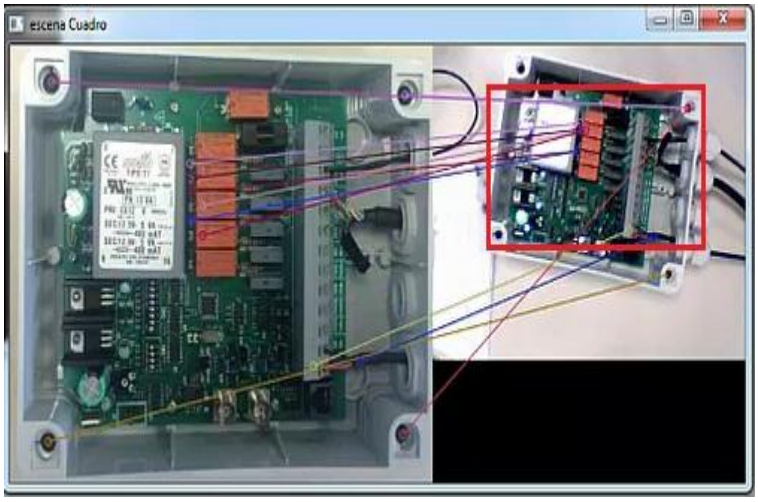


Figura 5-12: Error de contorno. Ejemplo C.

6 CONCLUSIONES Y TRABAJOS FUTUROS

La vida es el arte
de sacar conclusiones suficientes
a partir de datos insuficientes.
-Samuel Butter-

En este último apartado del proyecto se presentan las conclusiones y un breve resumen de los resultados obtenidos, así como los problemas que se han tenido que resolver a lo largo del proceso. También se hace hincapié en las posibles líneas futuras de este área y las ampliaciones que se pueden realizar para dar continuidad a este trabajo.

6.1 Cumplimiento de Objetivos y Conclusiones

El objetivo principal de este proyecto era obtener, de un objeto capturado con una cámara, información sobre su traslación, rotación, escalado y sus coordenadas de profundidad en tiempo real. Otro objetivo era conseguir un programa que minimizase los tiempos de ejecución, buscar los algoritmos óptimos y obtener flexibilidad en cuanto a patrones modificables o ampliaciones futuras sobre la base de datos con la que se quiera trabajar. Ambos objetivos se han cumplido, buscando siempre soluciones para satisfacer los tiempos de cálculo. Para ello, se presentaron diversas opciones entre las que se eligió siempre la que mejor se amoldase a cada necesidad. Por último, es importante destacar que la programación de los algoritmos que se ejecutan utiliza funciones compatibles con OpenCV 3.0, la última versión disponible en el momento de realizar este trabajo, y el lenguaje de programación elegido fue C++. Se trata de un lenguaje multiplataforma, con capacidad para adaptarse fácilmente a entornos de trabajo diferentes.

Para lograr dichos objetivos se han estudiado diversos métodos de comparación entre imágenes, entre los que se seleccionaron los algoritmos de visión artificial SIFT y SURF. El algoritmo SURF obtuvo los mejores resultados en pruebas previas al desarrollo de la aplicación, por lo que éste fue nuestro principal candidato. A pesar de tener cierta complejidad de representación, un gran peso computacional y requerir extraer las características de nuestras imágenes de la base de datos, antes de iniciar la aplicación, hemos conseguido resultados bastante buenos, con un rendimiento aceptable. Otra ventaja de este método es su potencial para encontrar similitudes que no son triviales para otros algoritmos y mucho menos a simple vista. Por todos estos motivos, SURF ha sido el algoritmo utilizado en este proyecto.

Sin embargo, ocasionalmente se han producido una serie de deficiencias que no permitían alcanzar la solución esperada, como ocurría en las siguientes situaciones:

- Las imágenes con cambios bruscos de iluminación o que contienen exceso de ruido provocan que la extracción de características sea errónea, debido a las limitaciones de la transformada SURF, pese a que ésta sea menos sensible a estos cambios que su antecesor, el algoritmo de SIFT. Este tipo de situaciones pueden impedir, a veces, el reconocimiento del objeto.

- En imágenes con un escaso número de puntos característicos o pocas correspondencias entre ellas es más complicado realizar una asociación correcta y pueden producirse problemas en el reconocimiento.

- El contorno obtenido mediante un modelo de rectas hacía difícil encontrar la zona de unión que encuadrase la figura en su interior de manera rectangular, siendo a su vez sensible a cambios de rotación, la creación directa de un rectángulo, ya que era frecuente que el objeto se saliera parcialmente del contorno delimitado.

Las posibles mejoras o soluciones que se desarrollaron en el proyecto se centraron en las deficiencias que hemos citado anteriormente, intentando mejorar cada una de ellas:

- La selección de imágenes y el espacio de captura del vídeo: se consiguen mejores emparejamientos para el cálculo de la homografía fijando distintos valores para los parámetros de distancia mínima y máxima entre puntos característicos, eligiendo el intervalo en el que mejores resultados se obtienen.

- El trabajo de comparación de imágenes con escaso número de puntos característicos: se soluciona con la incorporación de filtros y algoritmos de optimización que exigen un mínimo de emparejamientos entre la imagen capturada y la imagen del patrón para considerar ambas afines.

- El contorno: se soluciona con el requisito indispensable de perpendicularidad entre las líneas que unían las esquinas del objeto. Dicha perpendicularidad se consigue imponiendo la misma pendiente entre lados opuestos del polígono que forman las cuatro líneas.

Por último, cabe destacar que los resultados obtenidos dependen del conjunto de imágenes proporcionado y de su calidad, pudiendo ofrecer, desde soluciones casi óptimas con una cámara de resolución media de 320 x 240 pixeles, a otras que no son las más deseables, si conseguimos capturas con exceso de ruido o de alta resolución, que ralentizarán notablemente el rendimiento del programa.

6.2 Líneas Futuras

A continuación se detallan aspectos que se podrían mejorar en versiones futuras de este mismo algoritmo, para implementar aplicaciones de Reconstrucción 3D y Realidad Aumentada, con mayores requerimientos computacionales.

En la actualidad, las características y descriptores de imágenes son fundamentales y tienen muchas aplicaciones que van, desde el reconocimiento y seguimiento de objetos en movimiento, hasta la navegación de robots de forma autónoma. En este trabajo se presenta especial interés en otra aplicación relevante, como es obtener las coordenadas 3D de un objeto: Realidad Aumentada.

Estas matrices de coordenadas deben cargarse en otra librería llamada OpenGL para conseguir la matriz de proyección. Pasaríamos de coordenadas del mundo 3D a coordenadas de pantalla en 2D en nuestro plano de proyección.

OpenGL es una librería gráfica que permite la manipulación de gráficos 3D a todos los niveles. Puede usarse bajo todo tipo de SSOO e incluso usando una gran variedad de lenguajes de programación. Tiene un gran potencial y sus aplicaciones no paran de crecer de forma exponencial.

En una ampliación del proyecto se podrían insertar en el campo de visión de la cámara modelos 3D elaborados con OpenGL que fuesen sensibles a los movimientos de traslación y rotación, consiguiendo mediante un par de cámaras esteroscópicas la información instantánea proporcionada por las matrices de coordenadas de profundidad. Esto podría desarrollarse con vista a proyectos de distintos ámbitos como:

- La industria, facilitando las fases de montaje, reconociendo el estado actual e indicando por realidad aumentada el siguiente paso a ejecutar por el técnico.
- La medicina, incorporando métodos de aprendizaje en aquellas actividades que siguen una serie de pasos encadenados.

- Video-juegos, combinando el campo de visión del mundo real con modelos insertados en el campo de visión del jugador, consiguiendo una interactividad y realismo superiores a la realidad virtual.
- Automovilismo, enseñanza, uso doméstico, turismo, etc.

Figura 6-1 Captura del proceso de montaje en realidad aumentada desarrollado por la compañía BWM. [22]

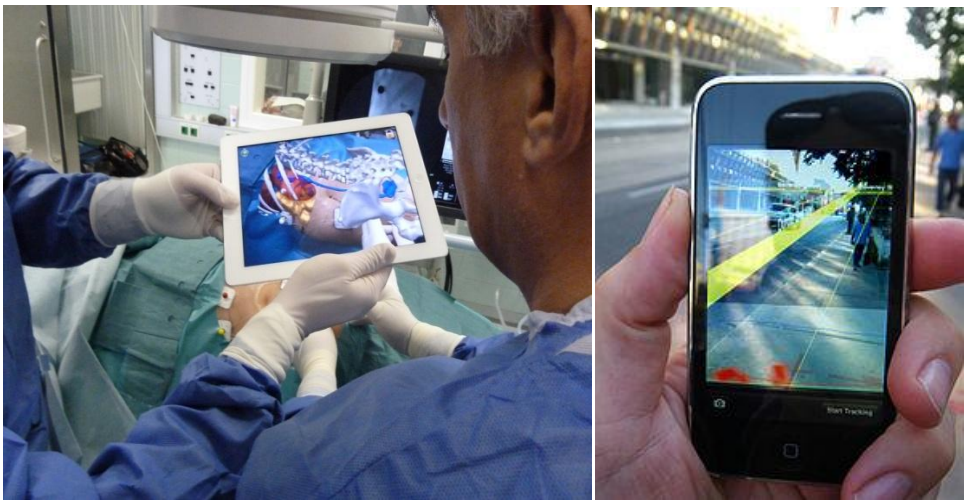


Figura 6-2 (izq.) Ejemplo de implementación de la Visión Artificial en el campo de la medicina.[23]

Figura 6-3 (dcha.) Ejemplo de implementación de la Visión Artificial en el teléfono móvil como complemento de un programa de uso del GPS.[24]

Figura 6-4 (izq.) Ejemplo de implementación de la Visión Artificial en el campo de los videojuegos.[25]

Figura 6-5 (dcha.) Ejemplo de implementación de la Visión Artificial en la domótica o el uso cotidiano del usuario.[26]

REFERENCIAS

[1] Wannes van der Mark and Dariu M. Gavrila. Real-time dense stereo for intelligent vehicles. IEEE Transanctions on intelligent transportation systems, 2006.

[2] Member Guofeng Zhang, Tien-Tsin Wong and Hujun Bao. Consistent depth maps recovery from a video sequence. IEEE, 2009.

[3] Luc Van Gool Andreas Ess, Bastian Leibe. Depth and appearance for mobile scene analysis. ICCV 2007.

[4] Johannes Kepler, "*Astronomiae pars Optica* ", 1604.

[5] "*Newton y el espectro de color"* . Disponible en: http://www.webexhibits.org/colorart/bh.html

[6] Hermann Ludwig Ferdinand Von Helmholtz, "*Handbuch der Physiologischen Optik", 1867.*

[7] Max Wertheimer, *Estudio sobre la ilusión del movimiento aparente* , 1911.

[8] "*WERTHEIMER, MAX (1880-1943*)". Disponible en: http://www.psicoactiva.com/biografia/max-wertheimer.htm

[9] Azriel Rosenfeld, *El análisis de imagen digital*. Berlín. Springer-Verlag ISBN 0-387-07579-8., 1976.

[10] González RC, Wintz P., *Procesamiento Digital de Imágenes*. Segunda edición. Addison-Wesley, 1987.

[11] Horn B. and Brooks M., *Shape from Shading*. The MIT Press, 1989.

[12] Horn B., *Robot Vision*. The MIT Press, 1986.

[13] Julesz, B., *Bell System Tech. J.*, 39, 1125 ,1960.

[14] Marr D., *Visión - Una Investigación Computacional en la Representación humana y procesamiento de la información visual. Freeman*, 1982.

[15]Documentación OpenCV. Disponible en: http://opencv-python-tutroals.readthedocs.org

[16] Chris Harris and Mike Stephens. "*A combined corner and edge detector"*. In Fourth Alvey Vision Conference, Manchester, 1988.

[17] Aracil López, Rafael. "*Desarrollo de un sistema cognitivo de visión para la navegación robótica*", Universitat Politècnica de València (pp. 18-24), 2012.

[18] Pablo Flores, Juan Braun. *"Algoritmo RANSAC: fundamento teórico"*, 2011.

[19] I. García Barquero, P. Sánchez-González , M. Luna Serrano, E.J. Gómez Aguilera1, "*Comparación de algoritmos detectores de puntos singulares para reconocimiento de objetos en vídeo quirúrgico"*. Grupo de Bioingeniería y Telemedicina (Universidad Politécnica de Madrid) y Centro de Investigación Biomédica en Red en Bioingeniería, Biomateriales y Nanomedicina, Zaragoza, 2012. (Figura 1).

[20] Figura 2-8, fuente: http://img.directindustry.com/images_di/photo-g/smart-camera-machine-vision-rugged-compact-17132-5875439.jpg

[21] Figura 6-1, fuente: https://es.wikipedia.org/wiki/RANSAC#/media/File:Fitted_line.svg

[22] Figura 6-1, fuente: http://mocadele.net/wp-content/uploads/2013/02/InstruccionAR.png

[23] Figura 6-2, fuente: http://www.augmentedrealitytrends.com/wp-content/uploads/2014/09/Augmented-Reality-in-Healthcare.jpg

[24] Figura 6-3, fuente: https://upload.wikimedia.org/wikipedia/commons/thumb/7/7b/MediatedReality_on_iPhone2009_07_13_21_33_39.jpg/230px-MediatedReality_on_iPhone2009_07_13_21_33_39.jpg

[25] Figura 6-4, fuente: http://i.telegraph.co.uk/multimedia/archive/02063/aug_2063679i.jpg

[26] Figura 6-5, fuentehttp://www.tocainteractivo.com/assets/images/realidad/realidad_4.jpg

Durante toda la elaboración del documento se han consultado las siguientes fuentes:

- Alejandro Madruga González, *" Inteligencia Artificial, el futuro del hombre",2013.*
- Robert Laganière*, "OpenCV Computer Vision Application Programming", 2014.*
- Tutoriales *OpenCV*: http://docs.opencv.org/modules/

- Foro *Stack Over Flow* : http://stackoverflow.com/questions/

- Foro *La Web del Programador*: http://www.lawebdelprogramador.com/foros/

- Foro *Club del Phi*: http://www.clubdelphi.com/foros/

- Empresa *Jasvisio*: http://www.jasvisio.com/team-es.html

- Web de matemáticas *Stack Exchange*: http://math.stackexchange.com/