Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería Electrónica y Automática

DESARROLLO DE UN SISTEMA DE VISIÓN PARA LA MONITORIZACIÓN DE OPERACIONES MANUALES

Autor:

Cantera Pérez, Rodrigo

Tutor:

Gómez García-Bermejo, Jaime
Departamento de Ingeniería de Sistemas y Automática

Co-Tutor:

Zalama Casanova, Eduardo
Departamento de Ingeniería de Sistemas y Automática

Valladolid, Octubre 2015

Resumen

En el presente trabajo de fin de grado se llevará a cabo el desarrollo de un sistema para el seguimiento de operaciones manuales de ensamblaje industrial haciendo uso de múltiples técnicas tales como visión estéreo, estimación de posición mediante marcadores y reconocimiento de objetos.

El sistema permite reconocer las herramientas utilizadas y estimar su posición y orientación en el espacio sirviendo de apoyo al operario y obteniendo datos que posteriormente puedan ser tratados para determinar si la operación o la forma de llevarla a cabo pueden ser mejoradas. Esto se traduce en un aumento de la eficiencia y permite reducir errores y riesgos.

Dado que la naturaleza y características de las operaciones a monitorizar puede pueden ser de naturalezas muy diversas, el desarrollo del sistema se ha planteado de tal manera que este pueda ser modificable y adaptable con facilidad a las características de cada situación, Sirviendo además como base para futuros trabajos.

Por ello, todas las herramientas utilizadas en su creación tienen la característica común de ser de fácil adquisición y contar con una amplia documentación. El sistema se ha programado en Microsoft Visual Studio 2013 para Windows 7 utilizando OpenCV como librería de visión por computador y webcams de consumo como elementos de captura.

Palabras Clave

Visión 3D, Visión estéreo, Estereopsis, OpenCV, CUDA, Monitorización de operaciones, Reconocimiento de objetos

AGRADECIMIENTOS

A mis padres por todo el apoyo y por no perder nunca la fe, a mis abuelas por todas las velas que han gastado, a mis hermanos que son unos soletes y a mis amigos de siempre por las risas.

Índice General

Índice de figuras

Índice de tablas

1. Introducción

1.1 Marco del proyecto

A pesar del alto grado de automatización en el ámbito industrial, muchas veces es necesaria la intervención humana para llevar a cabo operaciones que un robot no es capaz de realizar, ya sea por la dificultad que entraña o por que no es rentable utilizar uno a tal efecto. Este trabajo intenta abordar algunos de los problemas derivados de este tipo de operaciones.

Las operaciones manuales de ensamblaje industrial son objeto de estudio por parte de la sección de ingeniería de las empresas, que definen de manera estricta los pasos y el orden en el que efectuarlos así como las herramientas a utilizar en cada momento. El objetivo de estos estudios es establecer una manera estándar de trabajar que sea eficiente y que minimice la posibilidad de cometer errores. No obstante es imposible predecir por completo el efecto que se produce al introducir el factor humano y normalmente estas operaciones cuentan con una tasa de errores superior a aquellas completamente automatizadas.

El control de calidad es una cuestión presente en la industria y que consume tiempo y recursos tanto materiales como humanos para detectar los elementos que no cumplen los requisitos establecidos. En este punto muchas veces resulta imposible revertir el error, teniendo que descartar el elemento defectuoso y asumir las pérdidas que esto supone. En otras ocasiones es posible llevar a cabo una reparación, lo que significa un tratamiento individualizado del elemento con el consiguiente consumo de tiempo y recursos.

El problema inherente al control de calidad es que se realiza cuando el elemento ya está fabricado, y cuando se detecta un elemento defectuoso es necesario revisar todo el sistema para determinar la causa del defecto. Además, la tasa de elementos que no superan los controles de calidad aumenta con relación a los procesos de fabricación automáticos, debido la propia condición humana. Por ello es necesario desarrollar un sistema que monitorice directamente la operación y que en caso de encontrar algún defecto permita efectuar una acción correctiva inmediata.

Actualmente, en las industrias donde se llevan a cabo operaciones manuales de ensamblado existe un encargado responsable del control de calidad de las operaciones. Esto puede ser motivo de tensión entre los trabajadores debido a la dificultad de determinar el origen del fallo, ya que es imposible que el encargado esté en todo momento al tanto de todas las acciones realizadas por los operarios a su cargo. La solución a este problema actualmente consiste en marcar cada producto con código que de alguna manera permite identificar la persona o personas que han trabajado en él, pudiendo determinar así el origen del fallo, pero para ello es necesario llevar a cabo un control de calidad a posteriori y eso es precisamente lo que se intenta evitar.

Sumado a los problemas anteriormente expuestos, en la industria es habitual sufrir fluctuaciones de volumen de pedidos que pueden exigir la contratación de nuevos empleados temporales. Este personal necesita una formación adecuada, para su rápida incorporación al puesto de trabajo, sin menoscabo de la calidad del proceso realizado.

Este trabajo explora la posibilidad de realizar un control de calidad en línea durante el propio proceso que permita subsanar errores in situ, y al mismo tiempo facilite el trabajo al operario sabiendo que dispone de un sistema que le guía y alerta cuando una operación no se realiza de forma correcta.

Como método para el registro de las actividades llevadas a cabo por el operario se propone el uso de visión tridimensional, también conocida como visión 3D. Se entiende como visión 3d la parte de la Visión Artificial que trata de comprender escenas del mundo real de forma automática, utilizando para ello, sus propiedades tridimensionales inferidas de imágenes digitales.

En trabajos anteriores [14] se ha utilizado un sensor Kinect (desarrollado por Microsoft para consola Xbox) para la obtención de estos datos. El funcionamiento del sensor Kinect se basa en la proyección de un patrón conocido e invisible al ojo humano a partir del cual se pueden inferir las características 3D de la escena.

Sin embargo, aunque se va a mantener la idea original de extraer información 3D, se sustituirá el sensor Kinect por la técnica más habitual, que es la visión estéreo. Este nuevo enfoque abre un amplio abanico de posibilidades que permite el uso tanto de hardware específico cómo hardware que se puede encontrar con facilidad y a precios asequible. Además se utilizará la librería de software libre Opencv que en los últimos años se ha convertido en un estándar en el diseño de sistemas de visión artificial y cuenta con una amplia comunidad que sirve de apoyo a desarrolladores y aficionados.

Estas características hacen de este proyecto algo fácilmente replicable, modificable y mejorable por cualquier persona con conocimientos de visión artificial ya que elimina la necesidad de aprender a utilizar y trabajar con el sensor Kinect y su suit de desarrollo y eleva la precisión en la detección ya que no se utiliza luz estructurada que es, una de las principales limitaciones al trabajar con Kinect [14]. Además dado que se usan materiales y herramientas con un elevado grado de compatibilidad, el resultado final será aplicable a multitud de entornos sin necesidad de ser modificado.

1.2 Objetivos

El objetivo de este trabajo se consiste en el desarrollo de un sistema que permita la monitorización de operaciones de ensamblado industrial como taladrado, pegado, atornillado, etc. con la menor interferencia posible, esto es, minimizando el uso de marcadores u otras herramientas que obliguen a alterar el entorno de trabajo de manera que afecte a la ergonomía y comodidad del trabajador.

Además dado que la tarea del control de operaciones se puede presentar bajo condiciones muy diversas, se quiere obtener una base a partir de la cual otras personas puedan trabajar y modificar según sus necesidades.

Para ello será necesario lograr ciertos hitos que se van a marcar de la siguiente forma:

- Preparación del entorno de programación, incluyendo la elección de la librería a utilizar y cómo adaptarla las necesidades del proyecto.
- Elección de las cámaras.
- Análisis de diferentes técnicas dentro del campo de la visión por computador que puedan ayudar a conseguir el objetivo y elección de las más adecuadas para hacerlo.
- Implementación de las técnicas escogidas teniendo en cuenta la forma más óptima llevarlo a cabo.
- Experimentación y validación de los resultados obtenidos.

1.3 Estructura del documento

Tras este capítulo introductorio se aborda una de las partes más importantes de este trabajo: la visión estéreo. Posteriormente se describen en detalle algunas de las técnicas más utilizadas en el ámbito de la visión por computador.

Una vez establecidas las bases teóricas, en el capítulo 3 se trata todo lo relacionado con el entorno de programación, desglosando las herramientas utilizadas e indicando la manera de utilizarlas correctamente.

La elección de las cámaras a utilizar no es una decisión trivial ya que determinan en gran medida la precisión del sistema y una elección correcta puede facilitar mucho el trabajo que se realice posteriormente. Es por ello que el capítulo 4 está dedicado en su totalidad a las cámaras.

En el capítulo 5 se encuentra la información relacionada con el espacio físico donde se llevan a cabo las operaciones de ensamblaje y cómo este ha sido replicado para maximizar el parecido de las pruebas con la realidad. Teniendo estas características en cuenta se estudia la mejor distribución posible para las cámaras así como las características físicas de los marcadores a utilizar.

En este punto las condiciones y requerimientos a cumplir están suficientemente claros y por ello en el capítulo 6 se habla del diseño e implementación del algoritmo de trabajo del software.

En el capítulo 7 se encuentran las pruebas realizadas para determinar la viabilidad y efectividad de los algoritmos propuestos así como la justificación de su elección en función de los resultados obtenidos.

En el capítulo 8 se detalla la manera de utilizar el sistema desarrollado, así como la forma de llevar a cabo la calibración inicial necesaria para su correcto funcionamiento.

Por último, en el capítulo 9 se hace un estudio económico del coste de implementación y desarrollo y en el capítulo 10 se exponen los resultados obtenidos así como posibles vías de trabajo a explorar en el futuro.

2. Fundamentos Teóricos

En este capítulo se explicará la teoría necesaria para poder comprender los métodos utilizados durante el desarrollo de la aplicación. Se explicarán sus características, puntos fuertes y debilidades y se justificará su elección.

2.1 Visión Estéreo

La visión estéreo consiste en la extracción de información 3D desde imágenes digitales, esta información se obtiene comparando la información de una escena desde dos puntos diferentes, para ello, se utilizan 2 cámaras desplazadas horizontalmente una respecto a otra, aunque son posibles más configuraciones *(Figura 2.1)*.

Figura 2. 1: Cámara estéreo comercial

Comparando las imágenes provenientes de cada cámara, se puede obtener la información de la profundidad en forma de disparidades, que son inversamente proporcionales a la diferencia entre las distancias de los objetos. Es decir, cuanto más lejos esté un objeto de ambas cámaras, menos diferencia habrá entre ambas imágenes *(Figura 2.2)*.

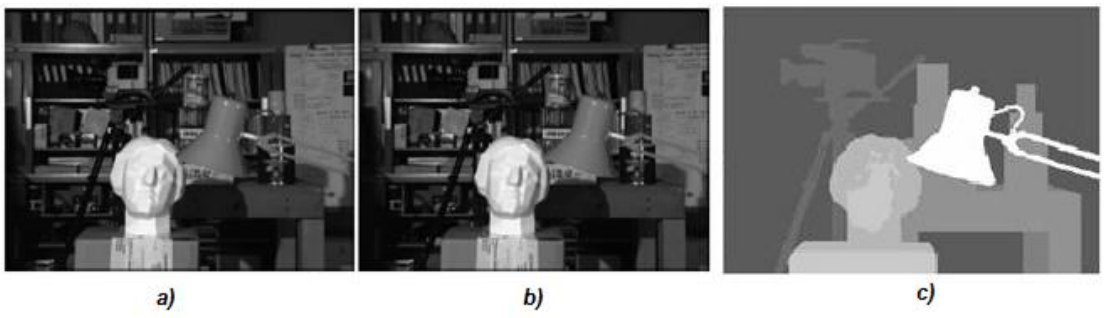


Figura 2. 2: a) Imagen de la cámara izquierda, b) Cámara derecha, c) Mapa de disparidad

2.1.1 Rectificación

El objetivo final de un sistema estéreo es obtener un mapa de profundidades que refleje la escena real. Como ya se ha visto para ello es necesario obtener dos imágenes desde dos puntos diferentes, y compararlas. Si se efectúa esta comparación sin ninguna consideración, vamos a obtener unos resultados lentos e imprecisos. Por eso es necesario tratar las imágenes antes y obtener información que facilite la tarea.

La rectificación de imágenes es un proceso de transformación que se usa para proyectar dos (o más) imágenes a un plano de imagen común. Este proceso cuenta con varios grados de libertad y existen multitud de estrategias para llevarlo a cabo. Su objetivo principal es el de simplificar el problema de encontrar los puntos correspondientes de varias imágenes como se muestra en la *(Figura 2.3).*

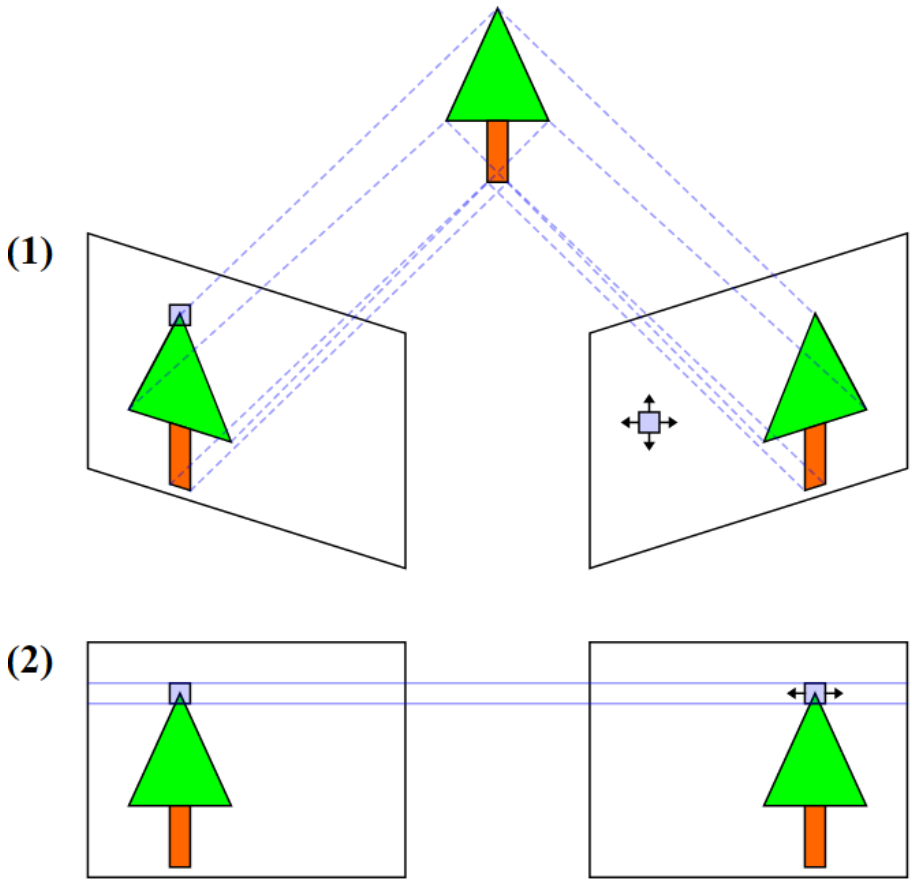


Figura 2. 3: Puntos equivalentes y rectas epipolares de una imagen

En la Mayoría de situaciones, encontrar correspondencias supone un problema de 2 dimensiones, sin embargo si ambas cámaras estuviesen perfectamente alineadas y fuesen coplanares, el problema se reduciría a una sola dimensión (una línea horizontal paralela a la línea que une las cámaras. La rectificación de imágenes se usa como alternativa a una alineación perfecta de las cámaras así como método de corrección de las posibles distorsiones inducidas por las mismas (distorsión geométrica). Los principales tipos de distorsión se pueden ver en la *Figura 2.4.*

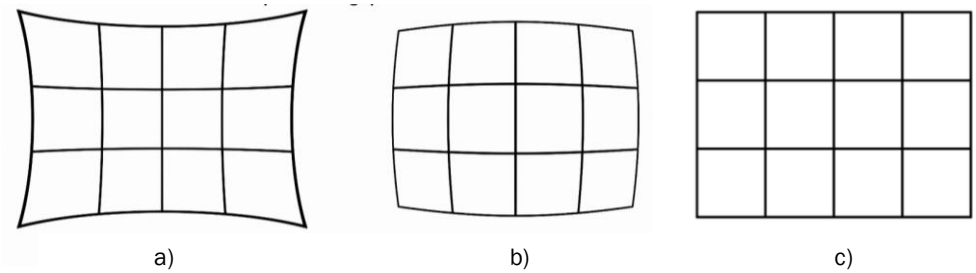


Figura 2. 4: Tipos de distorsión geométrica: a) Distorsión de acerico. b) Distorsión de barril. c) Sin distorsión

Más adelante se tratará el problema de la distorsión, de momento vamos a suponer que las imágenes que se necesitan rectificar están tomadas desde cámaras sin distorsión geométrica. La rectificación consistirá simplemente en una rotación que proyecte las imágenes a un mismo plano y una operación de escalado que las dote de un tamaño idéntico, obteniendo de esta forma dos imágenes con los pixeles correspondientes perfectamente alineados. (lo que se conoce como recta epipolar)

La relación entre la imagen original y la rectificada se representa mediante la matriz fundamental (si no se han calibrado las cámaras para eliminar las distorsiones) o matriz esencial (recoge los valores necesarios para eliminar las distorsiones y rectificar).

Las imágenes rectificadas deben cumplir las siguientes características:

- Todas las rectas epipolares son paralelas al eje horizontal
- Los puntos correspondientes tienen idénticas coordenadas verticales

Para comprender mejor esto se va a tratar el tema con el ejemplo de la *Figura 2.5.*

En azul podemos apreciar la escena y en verde y rojo ambas cámaras respectivamente, además en amarillo se representa la recta epipolar que une los puntos correspondientes en ambas imágenes.

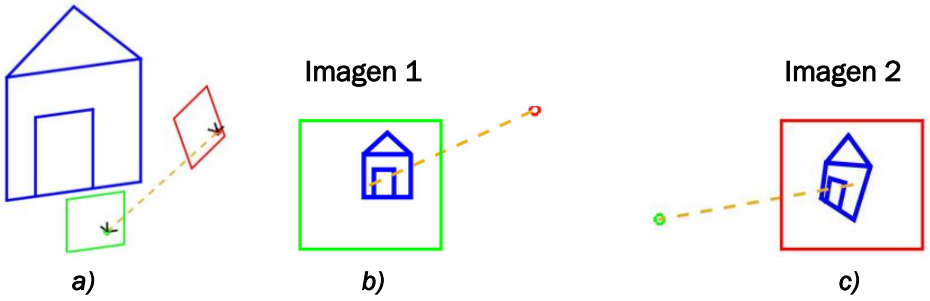


Figura 2. 5: a) Escena (azúl) vista desde dos localizaciones. b) Imagen cámara izquierda con su epipolo b) Imagen cámara derecha con su epipolo.

Así que va a ser necesario rectificarlas, para ello se efectuará una rotación un "retorcimiento" que permita ver la escena desde la misma perspectiva *(Figura 2.6).*

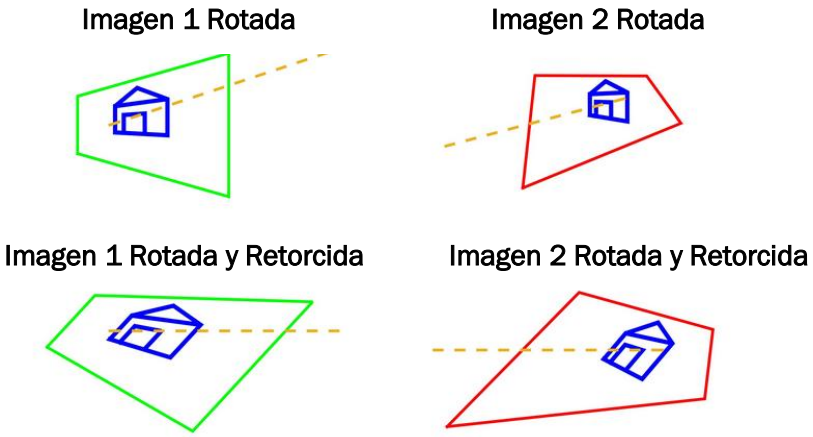


Figura 2. 6: Rotación y "retorcimiento" en una imagen

El resultado final son dos imágenes cuyos puntos correspondientes están alienados y que permiten calcular el mapa de profundidades *(Figura 2.7).*

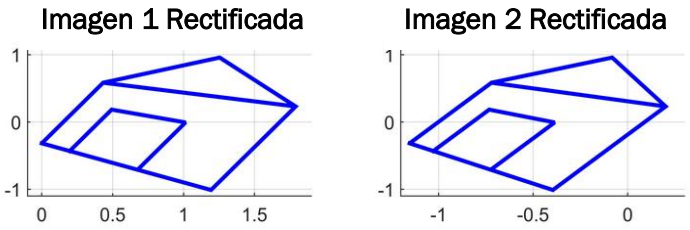


Figura 2. 7: Imágenes rectificadas

2.1.2 La línea base

Uno de los parámetros claves a la hora de diseñar un sistema de visión estéreo consiste en elegir la distancia de separación entre las lentes de las cámaras. Esta distancia se conoce habitualmente como la línea base *(Figura 2.8),* por su traducción directa del inglés (baseline).

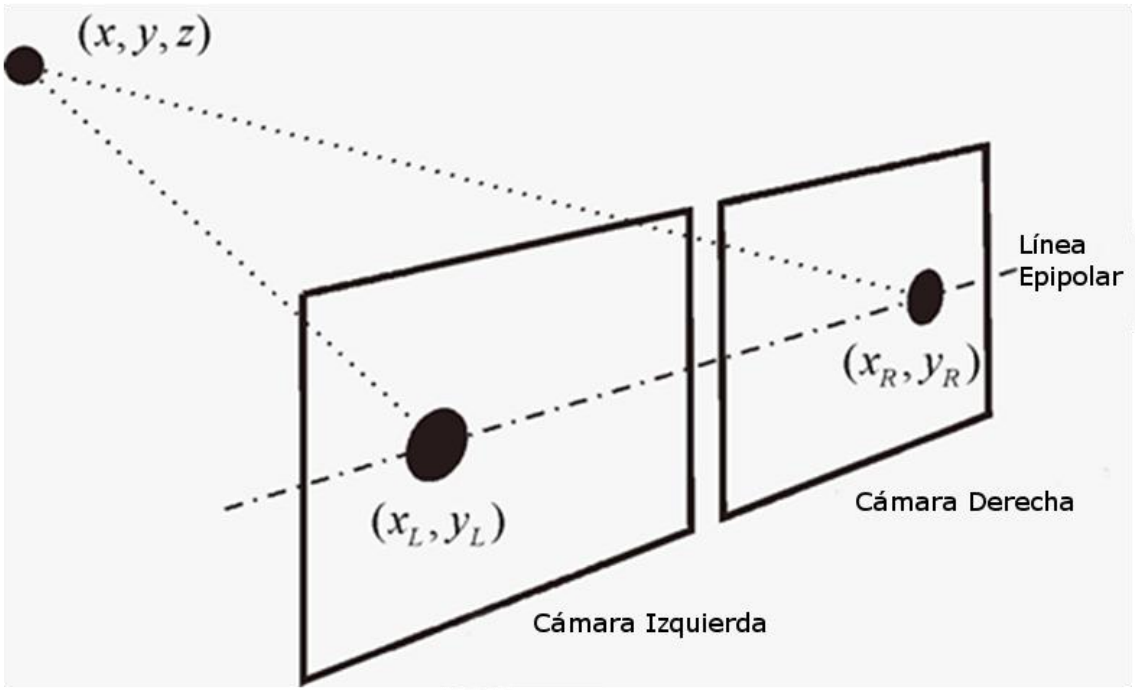


Figura 2. 8: Línea base de un par de cámaras

Si no se conoce ninguna característica concreta de la escena, normalmente se suele fijar una distancia igual a la separación que existe entre ojos humanos (65 mm). Cuando se utiliza dicha separación se considera que se está trabajando en ortoestéreo, sin embago también se puede utilizar una distancia más corta (hipestéreo) o superior (hiperestéreo), utilizándose estas dos cuando se quieren fotografiar objetos muy cercanos o muy distantes respectivamente.

Si por el contrario se conocen las dimensiones del espacio de trabajo, se usa la regla conocida como 1:30, indicando que la distancia de separación de las lentes tiene que ser 1:30 veces la distancia MINIMA al objeto más cercano [13].

Se define el área común de una escena como la zona que comparten ambas cámaras, esto significa que es la zona donde se podrá llevar una correlación estéreo efectiva y por lo tanto interesa que el área común coincida lo mejor posible con la región de la escena que nos interesa*(Figura 2.9).*

Figura 2. 9: a) Zona común de buen solapamiento. b) zona común con solapamiento escaso.

Utilizando la regla 1:30 se asegura que el sistema estéreo funciona bien para la zona del espacio en que se trabaje *(Figura 2.10).*

Figura 2. 10: Área común (recuadro interior) y zonas muertas (laterales) de una escena

2.2 Obtención del modelo de una cámara (Calibrado)

Calibrar una cámara es un paso indispensable en la visión por computador. Generalmente, esto se refiere al proceso de extraer los parámetros físicos de dicha cámara, por ejemplo el centro de la imagen, distancia focal, posición y orientación, etc. Es lo que se conoce como "calibración explicita" y es de uso universal en todos los aspectos de la visión por computador [6].

Cuando interesa predecir las coordenadas de la imagen partiendo de coordenadas conocidas de la escena (proyección) hace falta conocer más factores que los obtenidos con la calibración explicita, estos factores carecen de valor físico y su robustez no es importante mientras combinados puedan proveer una medición 3D correcta.

Una vez calibrada la cámara obtendremos:

Parámetros intrínsecos: se refieren a las características físicas de la cámara y engloban la distancia focal, el formato del sensor de imagen y el punto principal. (Matriz de la cámara. o Intrínseca)

Parámetros extrínsecos: denotan la transformación del sistema de coordenadas 3D del mundo real a las coordenadas 3D de la cámara. (Matriz Extrínseca).

2.2.1 Fundamentos Matemáticos

Ya se ha hablado antes de el concepto de distorsión que ocurre en las lentes de las cámaras, para corregirla, se tratan de manera independiente los factores radial y tangencial de la distorsión [17].

La corrección de la distorsión radial se puede expresar mediante la siguiente fórmula:

$$x_{corrected} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{corrected} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

Fórmula 4.1

De tal manera que para un pixel en las coordenadas ***(x,y)*** de la imagen de entrada, sus coordenadas en la imagen corregida de salida serán ***(Xcorrected, Ycorrected).*** Siendo k_1, k_2 y k_3 los parámetros de distorsión radial (ver Fórmula 4.3).

La distorsión tangencial ocurre debido que las lentes del objetivo no son perfectamente paralelas al plano en el que se toma la imagen, se puede corregir usando la Fórmula 4.2:

$$x_{corrected} = x + [2p_1xy + p_2(r^2 + 2x^2)]$$
$$y_{corrected} = y + [p_1(r^2 + 2y^2) + 2p_2xy]$$

Fórmula 4.2

Siendo p_1 y p_2 los parámetros de distorsión tangencial (ver fórmula 4.3)De esta forma se obtienen 5 parámetros que definen la distorsión de una cámara que se presentan en forma de una matriz de 1 fila y 5 columnas (Fórmula 4.4).

$$Distortion_{coefficients} = (k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3)$$

Fórmula 4.3

Ahora es necesario obtener la matriz intrínseca, para ello se utiliza la siguiente fórmula:

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Fórmula 4.4

La presencia de w se puede explicar debido al uso de un sistema de coordenadas homógrafo en el que ***w=z.***

Los parámetros desconocidos serán las distancias focales de la cámara *(fx fy)* y los puntos principales *(cx cy)* expresados en coordenadas de pixeles. Si ambos ejes comparten una misma distancia focal con un ratio de aspecto ***a***, entonces es posible dedicir que ***fy=fx*a***, obteniendo de esta manera una única distancia focal ***f***.

Los coeficientes de distorsión se mantendrán constantes independientemente de las resoluciones usadas al tomar las imágenes, pero los valores de la matriz intrínseca cambiaran si cambia la resolución así que será necesario calibrar el sistema de nuevo o escalarlos con la nueva resolución.

En la práctica, para obtener ambas matrices es necesario tomar imágenes de unos patrones conocidos como pueden ser un tablero de ajedrez, un patron de circulos simétricos o un patrón de círculos asimétricos. Cada vez que se obtiene una imagen y se encuentra dicho patrón, se obtiene una nueva ecuación para el cálculo de todos los valores antes mencionados.

En teoría basta con 2 imágenes de un tablero de ajedrez para poder resolver el sistema, pero en la práctica el ruido y otros factores hace que sea necesario unas 10 imágenes para obtener una buena calibración.

2.3 Segmentación

Segmentar consiste en separar las regiones que son de interés para el usuario de las que carecen del mismo. Como veremos más adelante esto se puede llevar a cabo de múltiples maneras o se puede utilizar una combinación de varias que ajuste más la zona segmentada a lo que se necesita.

Los algoritmos de segmentación se basan en los siguientes principios:

1. Discontinuidades del nivel de gris: consisten en segmentar la imagen a partir de los cambios grandes en los niveles de gris entre los píxeles. Las técnicas que utilizan las discontinuidades como base son la detección de líneas, de bordes, de puntos aislados ,etc.
2. Similitud de niveles de gris. es lo contrario al método anterior, las divisiones de la imagen se hacen agrupando los píxeles que tienen unas características similares. Algunas técnicas que usan esto pueden ser la umbralización o el crecimiento de regiones.

Hay muchas maneras y criterios mediante los cuales es posible segmentar una imagen, pero se va a hablar solamente de aquellos que se consideran útiles para el desarrollo del proyecto.

2.3.1 Umbralizacion

Normalmente los métodos del valor umbral "binarizan" la imagen de partida, es decir se construyen dos segmentos: el fondo de la imagen y los objetos buscados. La asignación de un pixel a uno de los dos segmentos (0 y 1) se consigue comparando su nivel de gris (g) con un cierto valor umbral preestablecido al que se llama umbral (u).

La imagen final (T) es muy sencilla y rápida de calcular ya que para cada pixel sólo hay que realizar una comparación numérica. La regla de cálculo correspondiente se puede ver en la fórmula4.5.

Formula 4.5 $T = 0$ si $g >= u$

$T = 1$ si $g < u$

Los métodos del valor umbral son métodos de segmentación completos, es decir cada pixel pertenece obligatoriamente a un segmento y sólo uno. Otros métodos de segmentación permiten que los segmentos se solapen. Si en la imagen existen varios objetos con una luminosidad similar, con un mismo tono de gris, todos los pixeles que los componen pertenecerán al mismo segmento *(Figura 2.11)*.

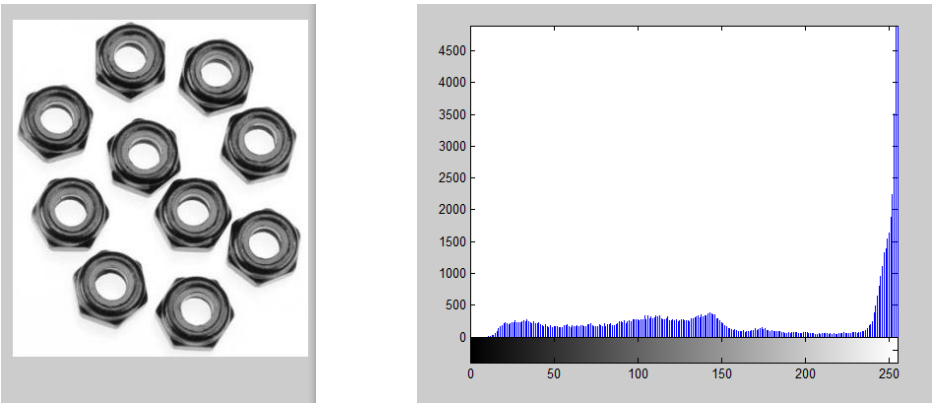


Figura 2. 11: Unas tuercas y su histograma

En la práctica siempre hay algún píxel que queda fuera del segmento aunque pertenezca al objeto, normalmente debido a ruidos en la imagen original. En función del valor umbral que se escoja el tamaño de los objetos irá oscilando *(Figura 2.12)*.

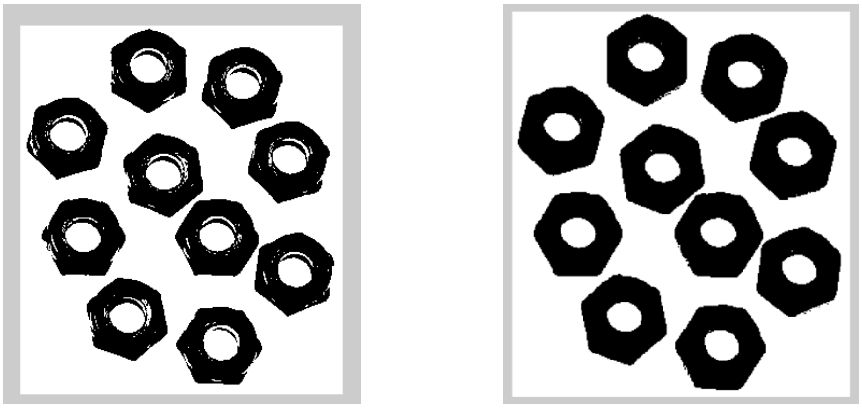


Figura 2. 12: Binarización y rellenado de huecos

Como se puede imaginar, la clave del éxito en estos métodos reside en una buena selección del valor umbral. Si las condiciones de operación (iluminación, parámetros de la cámara ,etc.) son constantes, este umbral se puede establecer empíricamente mediante pruebas y mantenerse constante en el tiempo.

Sin embargo en la realidad a veces no es posible trabajar bajo unas condiciones constantes y controladas, así que existen métodos para determinar el umbral de forma dinámica.

El método de Otsu, llamado así en honor a Nobuyuki Otsu [18] que lo inventó en 1979, utiliza técnicas estadísticas, para resolver el problema. En concreto, se utiliza la variancia, que es una medida de la dispersión de valores (en este caso se trata de la dispersión de los niveles de gris).

El método de Otsu calcula el valor umbral de forma que la dispersión dentro de cada segmento sea lo más pequeña posible, pero al mismo tiempo la dispersión sea lo más alta posible entre segmentos diferentes. Para ello se calcula el cociente entre ambas variancias y se busca un valor umbral para el que este cociente sea máximo *(Figura 2.13)*.

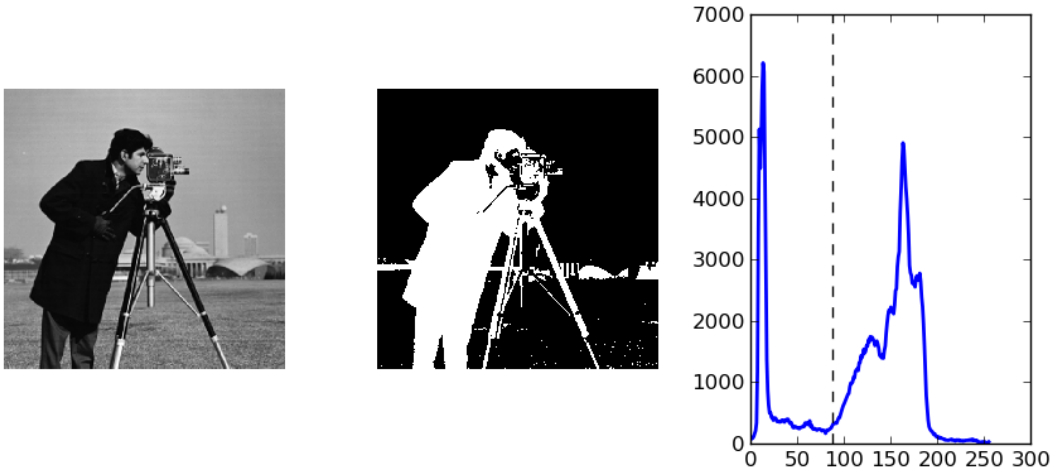


Figura 2. 13: Determinación de umbral según el método OTSU

Este tipo de métodos son muy sensibles a las variaciones en la luminosidad de la imagen.

Además del problema de la luminosidad, pueden aparecer otros problemas que se pueden reducir tratando la imagen antes de segmentar. A menudo se utilizan técnicas de reducción de la borrosidad o de incremento de la nitidez de los bordes.

Los métodos del valor umbral siempre utilizan información unidimensional de la imagen (normalmente un valor de intensidad o un valor de gris). No se tienen en cuenta otras informaciones, como por ejemplo los diferentes colores.

2.3.2 Discriminación por colores

La discriminación por colores se puede tratara como un caso general de la umbralización en escala de grises pero para imágenes a color.

Sabemos que las imágenes en color se componen de 3 capas de datos, correspondientes a los 3 canales de color con los que trabajaremos (Rojo, Verde y Azul, o RGB por sus siglas en inglés). De tal forma que es posible dividir una imagen en color en sus 3 planos de color *(Figura 2.14)*.

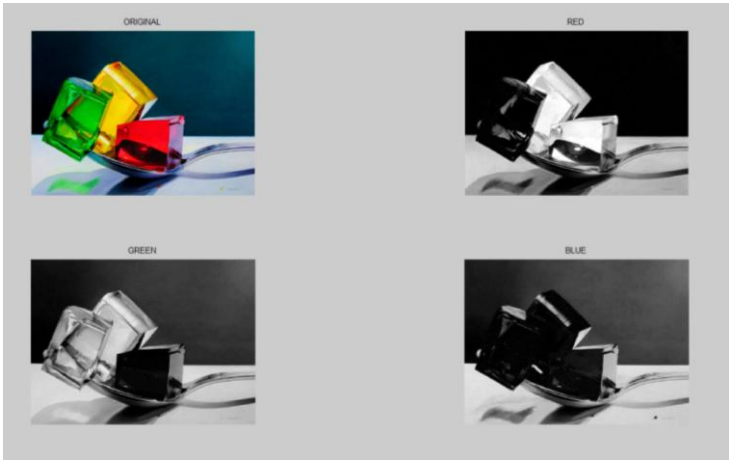


Figura 2. 14: Descomposición de una imagen en sus planos de color

Una vez se tiene la imagen dividida en sus 3 planos de color se puede llevar a cabo una umbralización en escala de grises independiente para cada plano, y el resultado final será la intersección de esas 3 umbralizaciones independientes *(Figura 2.15)*.

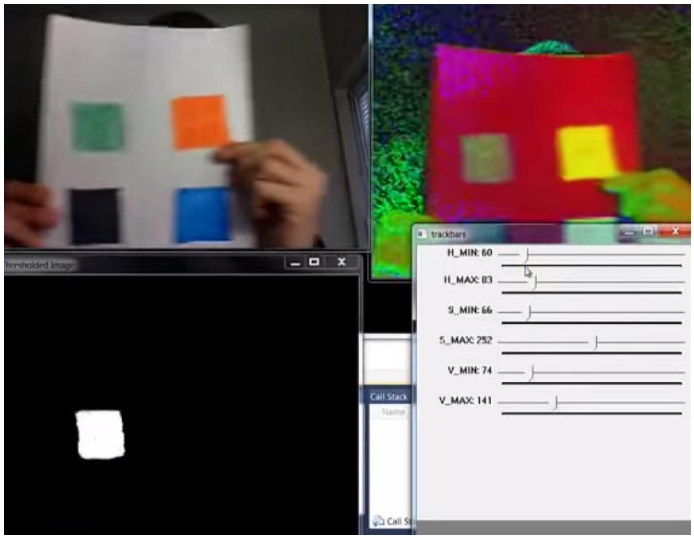


Figura 2. 15: Discriminación por colores

2.3.3 Establecimiento de regiones de interés

Una región de interés (ROI) es, como su propio nombre indica, una zona de la imagen en donde se quieren centrar los esfuerzos de la computación, considerando todo lo que quede fuera de la misma como no importante.

2.3.3.1 ROIS Estáticas

Cuando se trabaja bajo condiciones constantes y controladas, a veces es posible identificar parámetros que se sabe no cambiarán con el tiempo.

Por ejemplo, si en el sistema las cámaras siempre están en la misma posición, y en la escena aparecen zonas que no son de interés para la computación de la imagen, se pueden seleccionar las mismas de tal forma no se tomen en cuenta a la hora de procesar las imágenes.

Actualmente, en las cámaras profesionales es posible seleccionar las regiones de interés mediante el software del fabricante de tal forma que la cámara pase al sistema la imagen ya segmentada, reduciendo los tiempos de transmisión así como su volumen.

Además normalmente este tipo de cámaras permite seleccionar diferentes FPS para diferentes regiones, de tal forma que se puede obtener más información de las áreas que más nos interesan y actualizar la información de las menos importantes con menor frecuencia *(Figura 2.16)*.

Figura 2. 16: Distintas velocidades de adquisición en diferentes regiones de una cámara inteligente (Surveon 3 MP 30 FPS H.264 HDR Camera Series) [16]

Si no se dispone de una cámara con estas características la segmentación se tendrá que llevar a cabo una vez el sistema de computación haya recibido la imagen completa.

2.3.3.1 ROIS dinámicas

Se utilizan en caso de que se trabaje siempre con objetos de características muy similares pero que pueden estar situados en diferentes partes de la imagen.

Un ejemplo muy claro de este tipo de establecimiento de regiones de interés se puede encontrar en el reconocimiento de matrículas en vehículos.

Sabemos la forma aproximada que puede tener una matrícula, así que primero se efectuará un primer preprocesado de la imagen en busca de objetos parecidos, una vez se han detectado esos objetos se hace un procesamiento más exhaustivo de los mismos para identificar los números y letras que contienen *(Figura 2.17)*.

Figura 2. 17: Segmentación de regiones similares

2.3.4 Segmentación de objetos en movimiento

La segmentación de objetos en movimiento es un caso particular de la segmentación mediante regiones de interés dinámicas, pero se trata a parte ya que tendrá mucha importancia en el desarrollo del proyecto.

La idea básica detrás de este tipo de segmentación consiste en separar la acción principal del fondo. Para ello se compara la imagen actual con una imagen de referencia y se buscan las diferencias entre ambas, considerando las mismas como la acción principal.

Como se puede imaginar, la clave de este algoritmo consiste en la selección de una buena imagen de referencia. La aproximación más obvia consistiría en seleccionar una única imagen y comparar la actual con esa, sin embargo eso limita mucho la precisión y flexibilidad del sistema.

Lo que se hace entonces es elaborar una imagen de referencia que va evolucionando con el tiempo, de tal forma que si el fondo cambiase por cualquier motivo (la iluminación suele ser la mayor causa del cambio), la imagen de referencia se actualice y la detección siga funcionando correctamente *(Figura 2.18)*.

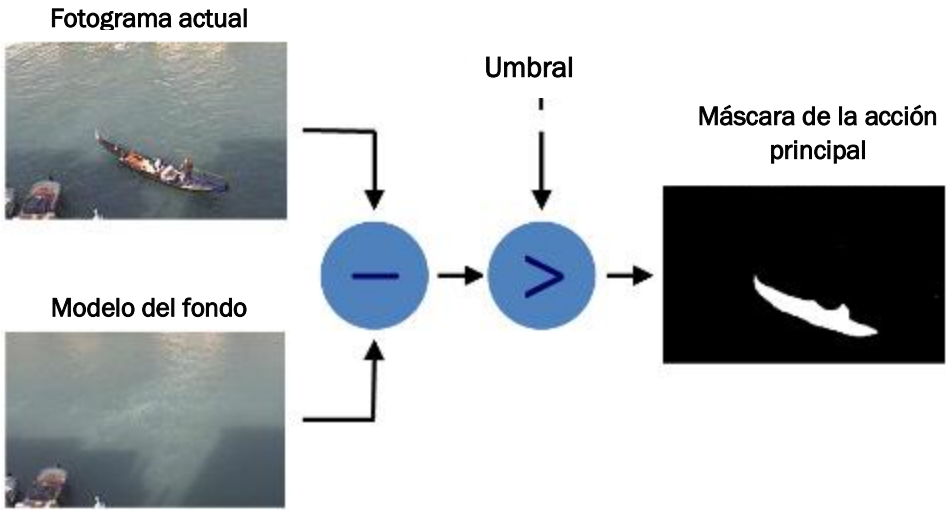


Figura 2. 18: Método para la segmentación del fondo y la acción principal

Para ello se utiliza como imagen de referencia una media de las últimas n imágenes capturadas por la cámara. Cuando ya se tiene esta imagen de referencia, lo único que queda por hacer es comparar la imagen actual con la de referencia *(Figura 2.19)*.

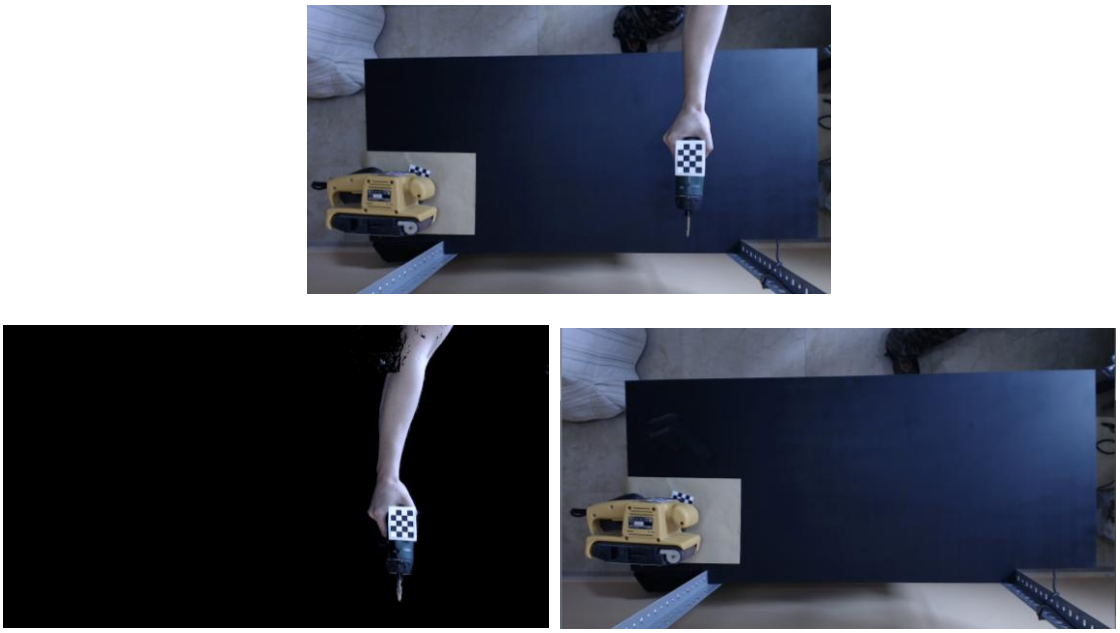


Figura 2. 19: Imagen completa (Arriba), Acción principal (Inferior izquierda) y Fondo (inferior derecha) segmentados

El coste computacional de este tipo de segmentación es bajísimo y permite operar en tiempo real con imágenes de alta resolución *(Figura 2.20)*.

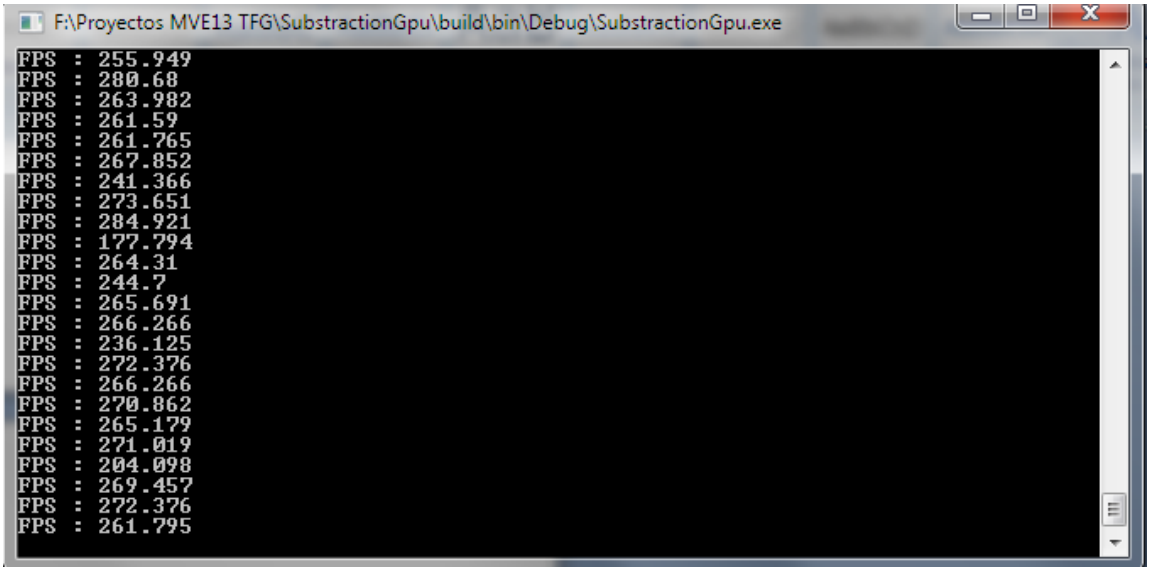


Figura 2. 20: Número de fotogramas que se pueden segmentar cada segundo

2.4 Detección de Características

Los humanos somos capaces de reconocer multitud de objetos con mucha facilidad, a pesar de que las imágenes que recibimos de los mismos pueden sufrir múltiples variaciones de tamaño, escala, o incluso de rotación y traslación. Somos capaces de reconocer objetos incluso cuando estos están parcialmente cubiertos o tapados por otros.

2.4.1 Tipos de características (rasgos):

No existe una definición oficial en cuanto a qué se considera un rasgo ya que esto depende mucho del tipo de aplicación, teniendo eso en cuenta se puede decir de forma generalista que un rasgo consiste en una parte "interesante" de la imagen.

Muchos algoritmos de visión por computador utilizan como punto de arranque detectores de rasgos por lo que se puede deducir que la característica más importante que debe tener un buen detector es la *repetitividad* (Si un mismo rasgo será detectado en dos o más imágenes diferentes de la misma escena).

Bordes: se consideran bordes aquellos puntos en los cuales existe una frontera entre dos regiones de la imagen. En general, un borde puede adoptar cualquier forma. En la práctica, se definen los bordes como aquellos puntos de la imagen que tienen un fuerte gradiente de magnitud *(Figura 2.21)*.

Figura 2. 21: Imagen original (Izquierda) y bordes detectados (Derecha)

Esquina / puntos de interés: ambos términos se usan muchas veces indistintamente debido a que ambos se refieren a rasgos puntuales de una imagen. el término "Esquina" surgió debido a que los primeros algoritmos se basaban en una detección de bordes y luego analizaban esos bordes en busca de cambios rápidos de dirección.

Dichos algoritmos evolucionaron posteriormente de forma que la detección de bordes no fuese necesaria. Fue entonces cuando se descubrió que dichos algoritmos detectaban también elementos de la imagen que no se consideran esquinas en el sentido tradicional de la palabra (un punto brillante sobre un fondo oscuro por ejemplo). Dichos puntos son conocidos como puntos de interés pero se usa frecuentemente el término esquina por tradición *(Figura 2.22)*.

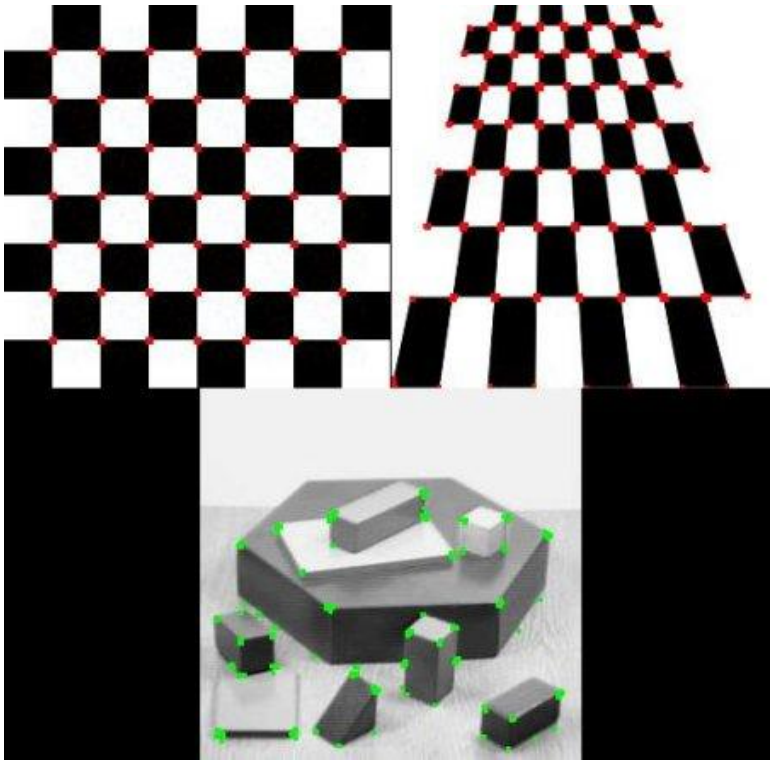


Figura 2. 22: Puntos de interés (esquinas) detectadas en diferentes imágenes

En general, un buen número de los algoritmos de detección de características se basan en la detección de esquinas ya que estas son invariantes a las rotaciones, lo que significa que se pueden encontrar los mismos puntos independientemente de si la imagen ha sido rotada o no. Sin embargo estos algoritmos no suelen ser invariantes frente al escalado (una esquina puede dejar de ser una esquina cuando la imagen es escalada)

2.4.2 SIFT

SIFT (Scale Invariant Feature Transform) es un algoritmo de detección de características propuesto por Lowe en 2004. Cuenta con 4 pasos principales [9] [10].

1 Detección de posibles extremos en el Espacio escalar DoG

La primera etapa sirve para identificar la localización y de los puntos clave, para ello se lleva a cabo una aproximación de LoG (Laplaciana de la Gussiana)conocida como DoG (Diferencia de Gaussianas). según la ecuación 2.1:

$$D(x, y, \sigma) = \big(G(x, y, k\sigma) - G(x, y, \sigma)\big) * I(x, y)$$

Ecuación 2.1

Donde *G* representa la función Gaussiana e *I* es la imagen. Como se puede deducir de la fórmula, ambas gausianas se relacionan entre sí mediante un parámetro *K* que actúa como relación entre ambos sigmas.

Esta operación da como resultado una lista de posibles candidatos a punto de interés. los cuales serán invariantes tanto en escala como en orientación

2 Localización de puntos clave

Además la diferencia Gaussiana también da un fuerte respuesta en los bordes, por lo que éstos deben ser eliminados. Para eso SIFT usa una matriz Hessiana para calcular las curvaturas principales, de tal forma que solo son interesantes aquellos puntos de interés para los ambos autovalores de la matriz Hessiana no difieren en un orden de magnitud o más; pues dichos puntos probablemente correspondan con bordes en la imagen y no con esquinas.

3 Asignación de orientaciones

Se toman los puntos vecinos en torno a cada punto de interés —en función de la escala— y se calcula la magnitud y dirección del gradiente. Entonces se hace un histograma de dichas direcciones ponderado por la magnitud del gradiente. El mayor pico en el histograma indica la orientación del punto de interés. Si existen otros picos por encima del 80% del más importante, se usan para crear otros puntos de interés en la misma posición y escala pero con diferente orientación.

4 Generación de descriptores

Para cada punto se toma un vecindario de 16x16 puntos. Este, a su vez, se divide en sub-bloques de tamaño 4x4 y para cada uno se crea un histograma de orientaciones. La concatenación en un vector de los valores de las cajas de cada histograma para los 16 sub-bloques del punto de interés constituye su descriptor.

2.4.3 SURF

El algoritmo de SURF está basado en los mismos principios y pasos que el SIFT , pero utiliza un esquema diferente y esto aporta una mayor rapidez [9][10]. Este algoritmo cuenta con las siguientes etapas:

1 Detección de puntos de interés

En vez de aproximar la LoG mediante DoG, se utiliza una aproximación por cuadrados, que es todavía más rápida que la anterior. Esta aproximación se puede convolucionar muy fácilmente con la imagen integral, pudiendo incluso calcularse en paralelo para diferentes escalas, lo que agiliza mucho el tratamiento *(Figura 2.23)*.

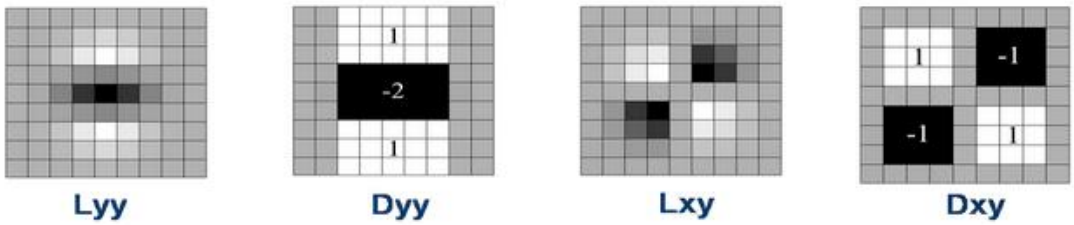


Figura 2. 23: Máscaras Laplacianas (Lyy, Lxy) y sus aproximaciones (Dyy, Dxy) [14]

Para la orientación, SURF utilizan la transformada de Haar [14] en una región circular que rodea a cada punto detectado. Escogiendo el elemento adecuado se hayan las orientaciones de cada punto de la región. Esta operación se lleva a cabo de manera muy rápida ya que una vez más calcular dichas convoluciones es barato computacionalmente hablando y se puede hacer en paralelo *(Figura 2.24)*.

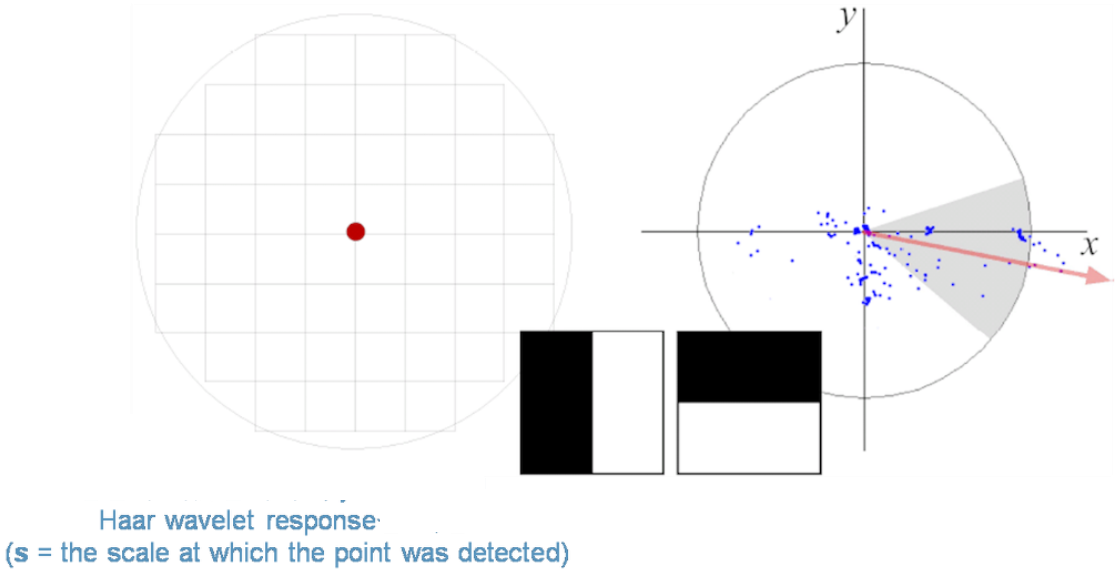


Figura 2. 24: Puntos detectados mediante la transformada de Haar [14]

La generación de descriptores se lleva a cabo de manera muy similar a SIFT

2.4.4 KAZE

KAZE [7].es un novedoso algoritmo de detección y descripción de características diseñado para trabajar en un espacio escalar no lineal. Hasta ahora los algoritmos que se han visto (SIFT Y SURF) se valen de construir o aproximar el espacio escalar Gausiano de una imagen, sin embargo, el desenfoque Gausiano no respeta las características originales de la imagen y desenfoca igualmente el ruido y los detalles que se consideran de interés, reduciéndose así la precisión de la localización

En contraste KAZE trabaja en un espacio escalar no lineal en 2D ya que utiliza un filtro de difusión no lineal, consiguiendo de esta manera que el desenfoque se adapte a los datos de la imagen, reduciendo el ruido pero manteniendo los detalles de la misma y obteniendo de esta forma una precisión superior *(Figura 2.25)*.

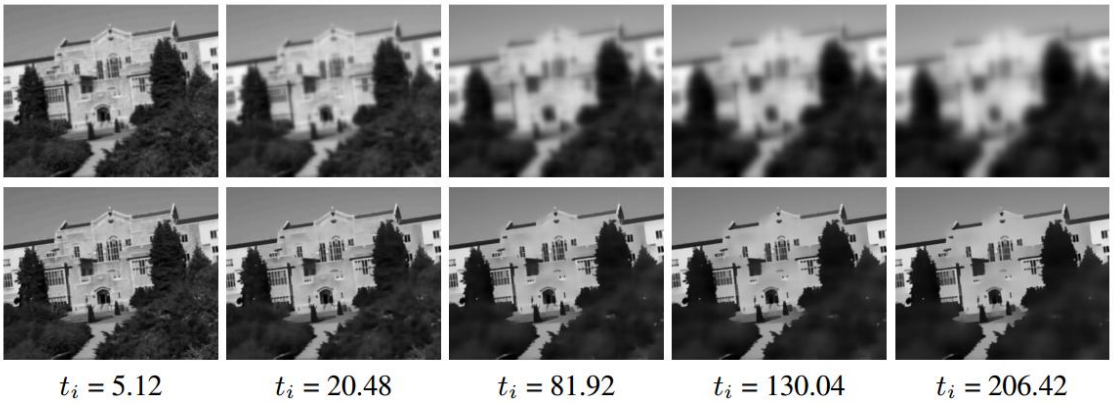


Figura 2. 25: Efecto de trabajar en un espacio escalar lineal (Fila superior) y no lineal (Fila inferior). Nótese que el desenfoque afecta solamente a los arbustos en el segundo caso [7]

En líneas generales, el algoritmo es más costoso computacionalmente hablando que el SURF debido a la construcción del espacio escalar no lineal, pero comparable al SIFT.

Gracias a que este algoritmo es capaz no solo de detectar si no de describir los puntos clave de una imagen, es posible obtener usar dicha descripción para encontrar puntos equivalentes entre dos imágenes *(Figura 2.26)*.

Figura 2. 26:Puntos equivalentes entre dos imágenes usando KAZE

2.5 Estimación de posición a través de elementos conocidos.

Establecer la posición y orientación de un objeto requiere cierto conocimiento del mismo o del entorno. Cuando no es posible o viable conseguir esa información, se recurre a técnicas como la visión estéreo de la que ya se ha hablado antes.

Teniendo un elemento cuyas características físicas o geométricas son conocidas con exactitud por el observador, es posible calcular la posición y orientación del mismo en función de cómo se está viendo en ese preciso momento eliminando la necesidad de una segunda cámara.

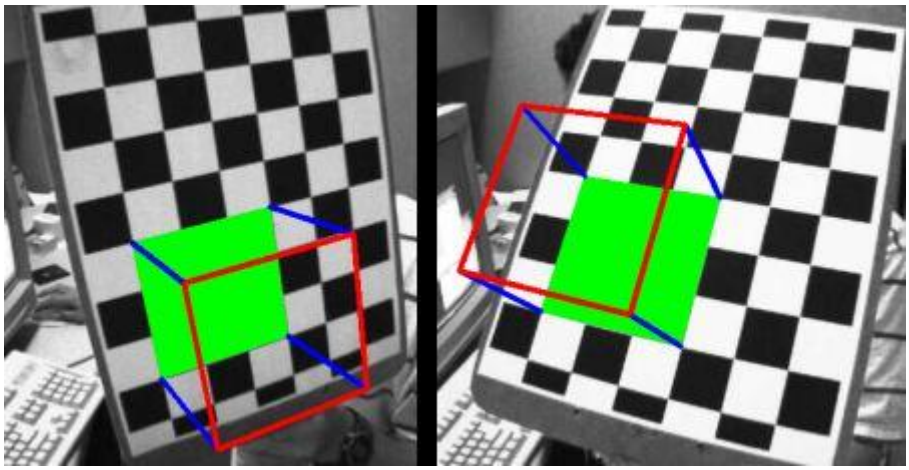


Figura 2. 27:Cubo dibujado sobre una imagen cuya posición y orientación han sido previamente detectadas

Esta forma de trabajar cuenta con los siguientes pasos:

<u>1º Detección de la plantilla en la imagen recibida:</u> supongamos que estamos utilizando una plantilla ajedrezada como la que se puede ver en la *(Figura 2.27)*.

Lo primero que hay que hacer para estimar la posición de la misma es comprobar que efectivamente la imagen recibida cuenta con una plantilla de las características indicadas, es decir, el algoritmo debe saber no sólo el tipo de plantilla utilizada si no también el número de cuadrados de alto y de ancho que tiene.

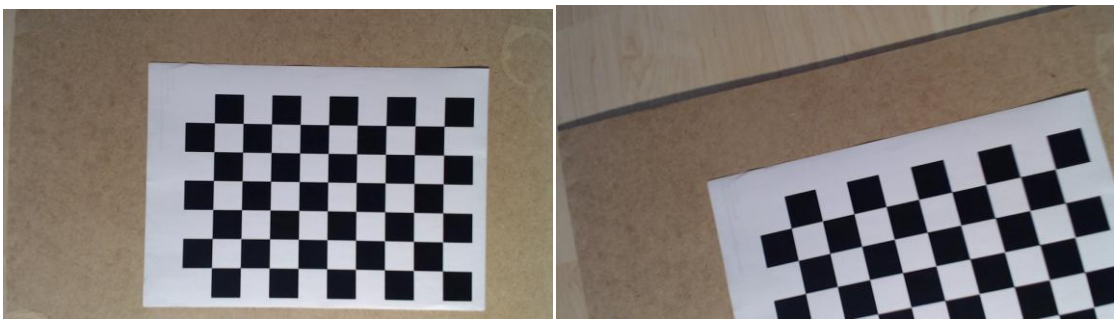


Figura 2. 28: Imagen de una plantilla válida (Izquierda) y no válida (Derecha)

De esta manera, aunque el algoritmo encuentre una plantilla cuadrada en la imagen, esta no será válida si no tiene las dimensiones esperadas (por ejemplo, aparece recortada) *(Figura 2.28)*.

La detección supone el cuello de botella del algoritmo ya que computacionalmente hablando es muy costoso determinar si la imagen recibida cuenta con un plantilla de las características adecuadas. Por ello, una vez se detecta una plantilla válida, el algoritmo designa una zona de la imagen cercana a la última detección de la plantilla donde empezar a buscar. Esto hace que las sucesivas detecciones se produzcan de manera muy rápida.

2º Detección de las esquinas de los cuadrados de la plantilla: Una vez identificado el área donde se puede encontrar la plantilla, se lleva a cabo una detección de esquinas basada en el detector de Harris [20] *(Figura 2.29)*.

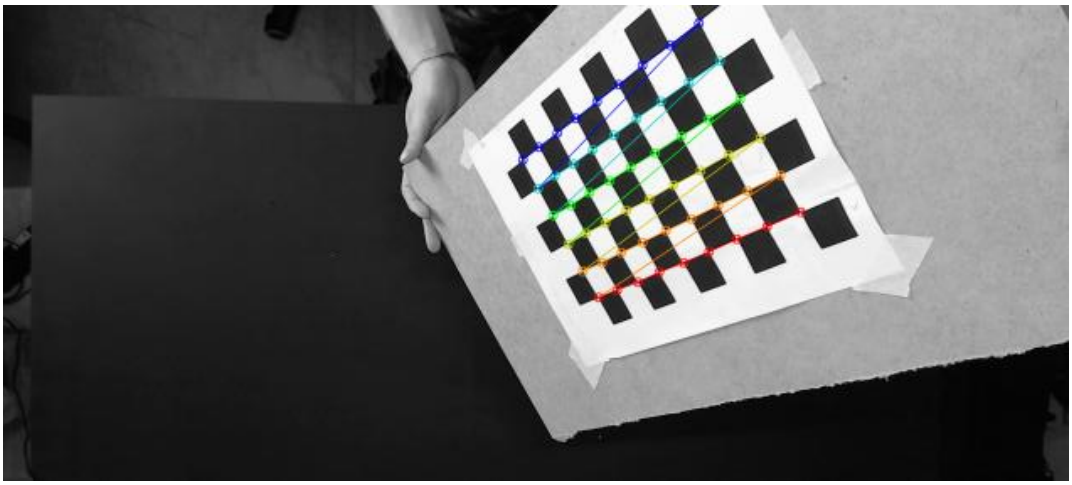


Figura 2. 29: Detección de esquinas en una plantilla

3º Determinación de la posición: Tener todas las esquinas localizadas permite estimar la posición de la plantilla mediante álgebra matricial.

2.6 CUDA

Los sistemas informáticos están pasando de realizar el "procesamiento central" en la CPU a realizar "coprocesamiento" repartido entre la CPU y la GPU.

Para posibilitar este nuevo paradigma computacional, se crea CUDA [19], una arquitectura de cálculo paralelo de NVIDIA que aprovecha la gran potencia de la GPU (unidad de procesamiento gráfico) para proporcionar un incremento extraordinario del rendimiento del sistema *(Figura 2.30)*.

CUDA intenta explotar las ventajas de las GPU frente a las CPU de propósito general utilizando el paralelismo que ofrecen sus múltiples núcleos, que permiten el lanzamiento de un altísimo número de hilos simultáneos. Por ello, si una aplicación está diseñada utilizando numerosos hilos que realizan tareas independientes (que es lo que hacen las GPU al procesar gráficos, su tarea natural), una GPU podrá ofrecer un gran rendimiento en campos que podrían ir desde la biología computacional a la criptografía o a la visión por computador, como es el caso.

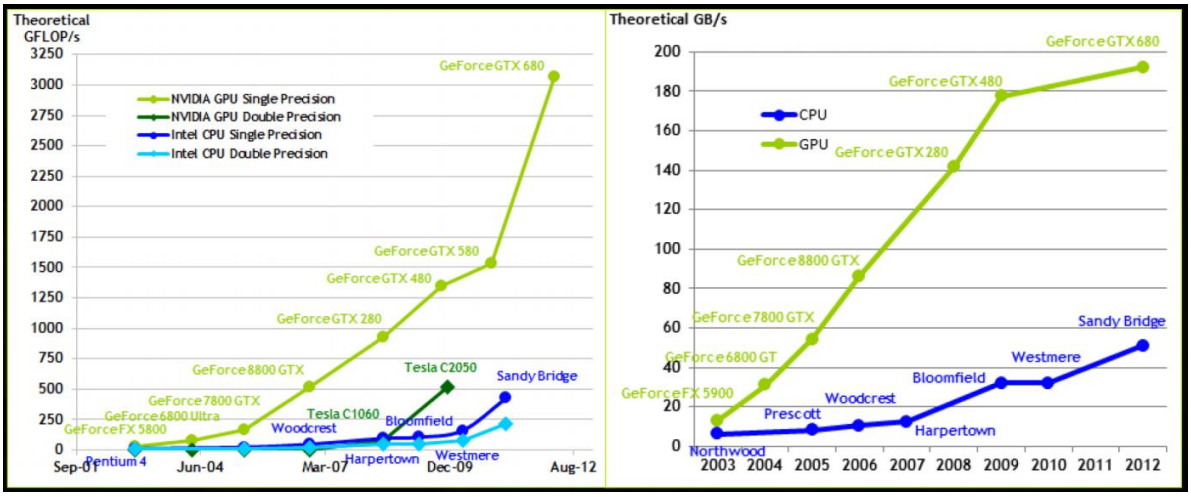


Figura 2. 30: Comparación de rendimiento entre CPUs y GPUs

Sin embargo hay que tener en cuenta que esta tecnología requiere transferir las imágenes entre la CPU y GPU, lo cual significa que existe un tiempo de retraso, así que no se debe suponer que todas las operaciones van a ser automáticamente más rápidas, hay ciertos casos para los que esto no se cumple.

CUDA fue originariamente concebida para mejorar el rendimiento de software de renderizado gráfico (Industria de los videojuegos), sin embargo es fácil darse cuenta que las tareas que se llevan a cabo en ese campo son altamente análogas a las que se usan en la visión por computador, solo que trabajando en sentido inverso (en vez de generar complejos gráficos, interesa analizarlos) *(Figura 2.31)*.

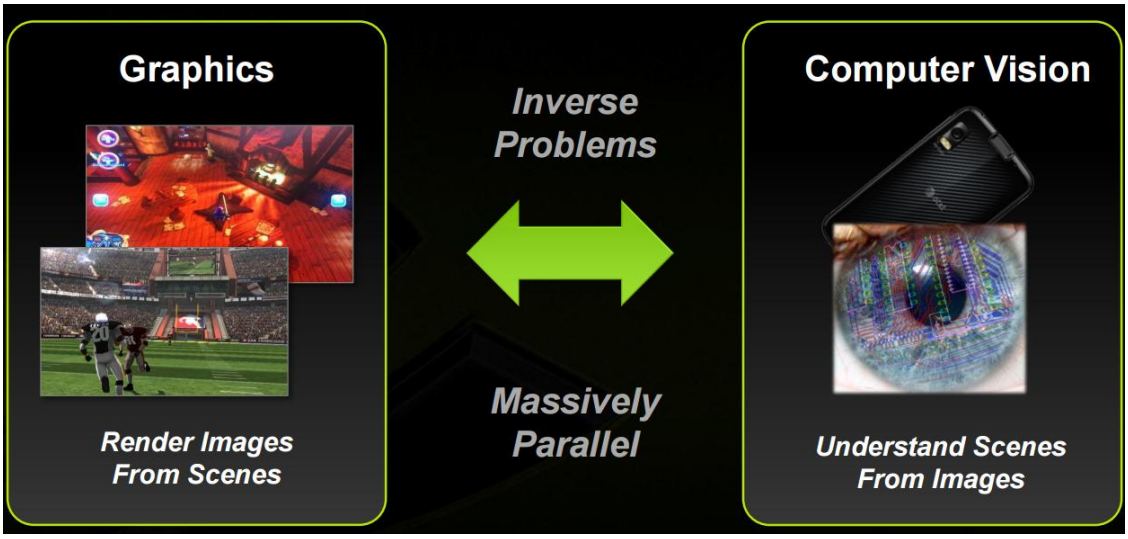


Figura 2. 31: Representación de la dualidad generación-análisis de imágenes [8]

Es por eso que esta tecnología ha tenido tanto éxito y aceptación en el campo de la visión por computador y supone un gran avance en cuanto a las posibilidades que esta ofrece.

OpenCV ofrece una integración con CUDA para algunas de sus funciones, de manera que somos capaces de utilizar la GPU del ordenador para llevar a cabo ciertas tareas que utilizando la CPU serían demasiados costosas o lentas [8].

Esta integración además se ha llevado a cabo de tal manera que permite un uso muy similar al que está acostumbrado el usuario de OpenCV, minimizando así el tiempo de aprendizaje y la facilidad de adaptación del código *(Figura 2.32)*.

```
Mat frame;
VideoCapture capture(camera);
cv::HOGDescriptor hog;

hog.setSVMDetector(cv::HOGDescriptor::
        getDefaultPeopleDetectorector());

capture >> frame;



vector<Rect> found;
hog.detectMultiScale(frame, found,
    1.4,  Size(8, 8), Size(0, 0), 1.05, 8);
```

```
Mat frame;
VideoCapture capture(camera);
cv::gpu::HOGDescriptor hog;

hog.setSVMDetector(cv::HOGDescriptor::
        getDefaultPeopleDetectorector());

capture >> frame;

GpuMat gpu_frame;
gpu_frame.upload(frame);

vector<Rect> found;
hog.detectMultiScale(gpu_frame, found,
    1.4, Size(8, 8), Size(0, 0), 1.05, 8);
```

Figura 2. 32: Similitudes de programación entre CPU y GPU

En resumidas cuentas podríamos decir que las ventajas y desventajas del uso de esta tecnología se pueden desglosar de la siguiente manera:

Ventajas:

- Código muy similar al de la CPU
- Idóneo para grandes operaciones paralelas y bajo flujo de datos entre CPU y GPU
- Incremento de velocidad significativo en muchas operaciones
- Hace que tareas como la visión estéreo o la detección de peatones sean realizables en tiempo real

Desventajas:

- Sólo 250 funciones
- Tipos de datos limitados: (escala de grises de 8 y 32 bits)
- Programación explicita para CUDA
- Sólo utilizable en GPUs de marca NVIDIA
- Hay operaciones que no aumentan su velocidad (Canny, por ejemplo)
- Tiene cierto retraso de encendido (esto afecta a aplicaciones que solo trabajen con una única imagen)

3. Herramientas de Desarrollo

En este capítulo se va a hablar de todas las herramientas utilizadas para el desarrollo del sistema utilizando como punto central la librería de visión artificial escogida (OpenCV) ya que en cierta manera define y acota el resto de opciones. Esto es debido a que la librería es el "cerebro" del sistema y el objetivo principal es conseguir que funcione de manera correcta.

3.1 Introducción a OpenCV

OpenCV [1] es una librería Open Source de visión por computador. Se creó con el objetivo de proponer una infraestructura común para las aplicaciones de visión por computador y para acelerar el uso de esta tecnología en productos comerciales.

La librería tiene más de 2500 algoritmos optimizados, lo que incluye tanto procedimientos clásicos como algunas de las técnicas más actuales. Estos algoritmos se pueden utilizar para detectar y reconocer caras, seguir el movimiento de los ojos de una persona, etc. Sin embargo en este caso su interés radica en que también es posible utilizarlos para la identificación de objetos, así como del seguimiento del movimiento de los mismos y producir nubes de puntos en 3D desde cámaras estéreo.

Se ha elegido trabajar con OpenCV debido a que está distribuida bajo licencia BSD, lo cual permite hacer libre uso de la misma tanto a particulares como empresas y a su compatibilidad con la mayoría de sistemas operativos actuales(Windows, Linux, Mac OS y Android), además siendo una librería enfocada a las aplicaciones de visión en tiempo real, puede hacer uso de las instrucciones MMX y SSE siempre que estén disponibles.

Actualmente se está empezando a dar soporte a CUDA y OpenCL, lo cual supondrá un rendimiento muy superior en sistemas que incorporen tarjetas gráficas compatibles con estas tecnologías.

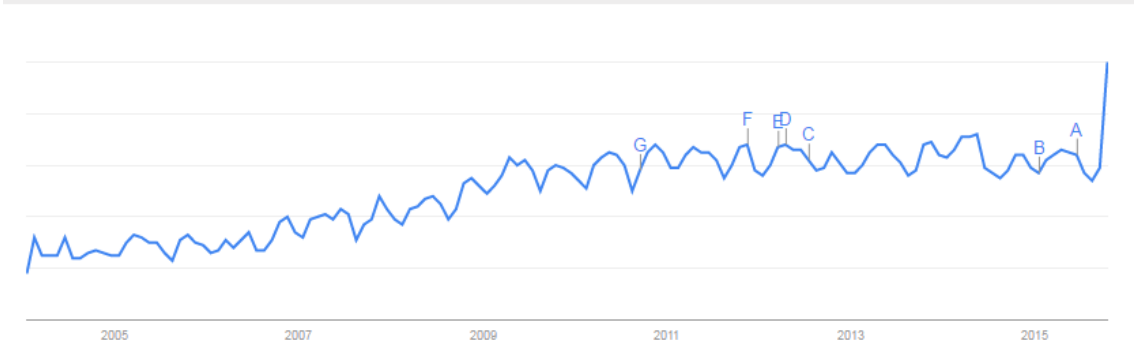


Figura 3. 1: Frecuencia de búsqueda del término "OpenCV" en los últimos 10 años

3.2 Instalación de OpenCV

En este proyecto vamos a utilizar Windows como sistema operativo así que sólo se cubrirá la instalación de OpenCV para este sistema, pero los pasos seguidos (con ligeras modificaciones) son válidos también en otros sistemas.

OpenCV ofrece un instalador automático, sin embargo utilizaremos directamente el repositorio para poder contar siempre con la versión más actualizada sin depender del lanzamiento oficial de versiones.

3.2.1 Clonación desde repositorio

Un repositorio es un sitio centralizado donde se almacena y mantiene información digital*(Figura 3.2)*.

La ventaja de trabajar con este método es que a parte de obtener la versión más actualizada de OpenCV, a la hora de compilar la librería podremos elegir qué módulos y características se quieren tener disponibles.

Para ello, nos dirigiremos a www.opencv.org, en el apartado "Downloads" elegiremos "Download from repository" y copiaremos la dirección de descarga

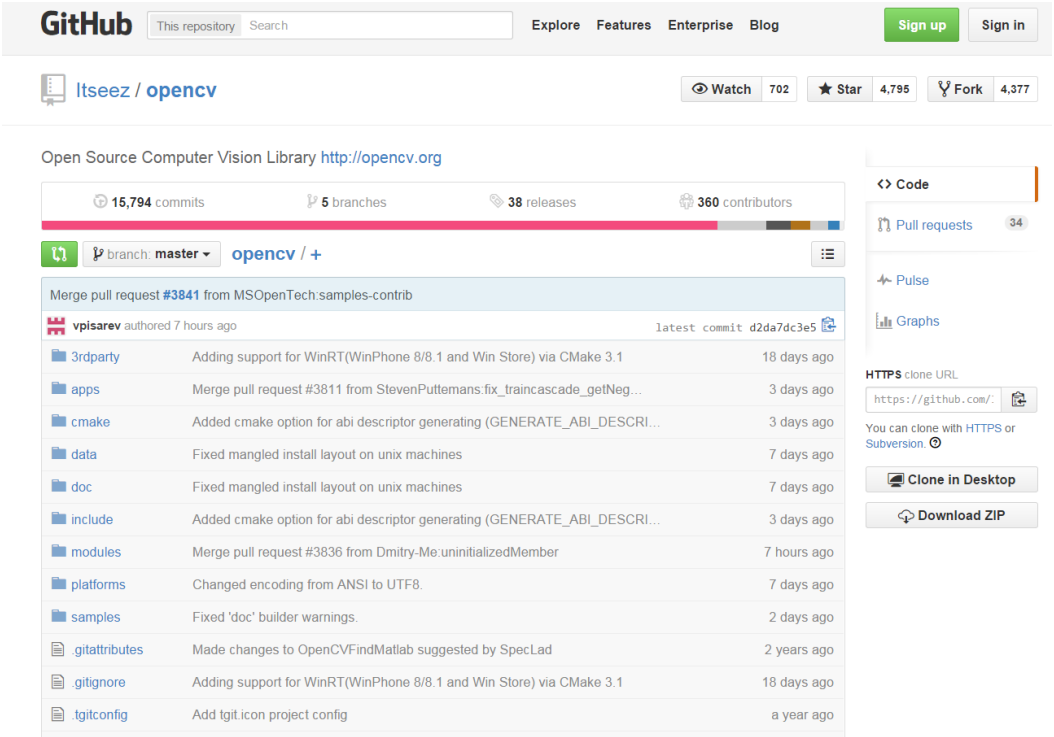


Figura 3. 2: Repositorio de OpenCV

Una vez tenemos la dirección, nos situaremos en la ubicación donde queremos clonar OpenCv *(Figura 3.3)* (en este caso en el directorio raíz del disco C), click derecho, GIT clone, pegar la dirección del repositorio de OpenCV y esperar a que se sincronice.

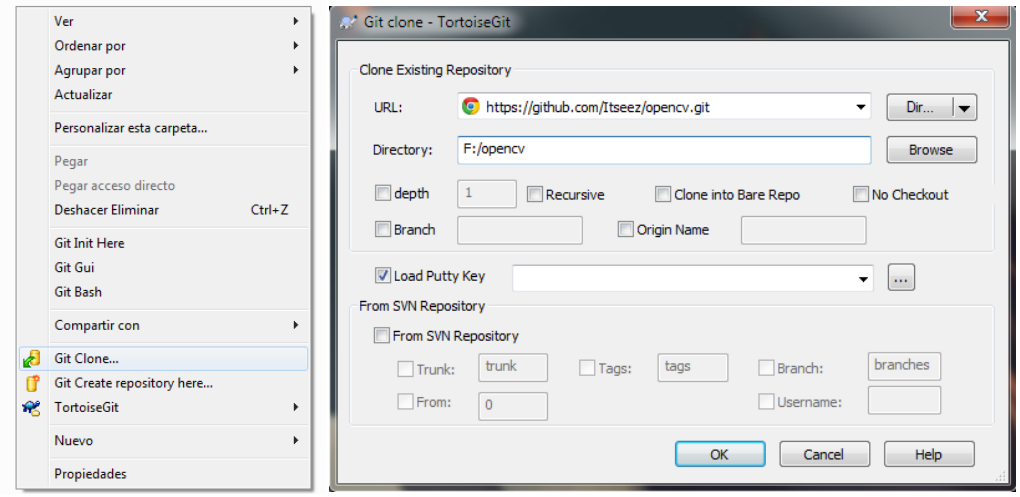


Figura 3. 3:Clonación desde repositorio

A partir de ese momento ya tenemos disponible el código fuente de la librería y siempre que queramos actualizarlo podremos seleccionar la carpeta del código fuente y efectuar un "Pull request"

3.2.1.1 GIT

Hay que entender que OpenCV cuenta con una alta participación de gente que constantemente está mejorando y añadiendo nuevas funcionalidades a la librería.

Eso hace necesario un sistema de control que homogenice los cambios que se producen dentro del proyecto, que controle posibles conflictos y que documente las modificaciones que va sufriendo el mismo a lo largo del tiempo *(Figura 3.4)*.

GIT es un sistema de control de versiones distribuidas, gratis y de código abierto, lo que le ha llevado a convertirse en un estándar en el control de desarrollo de software.

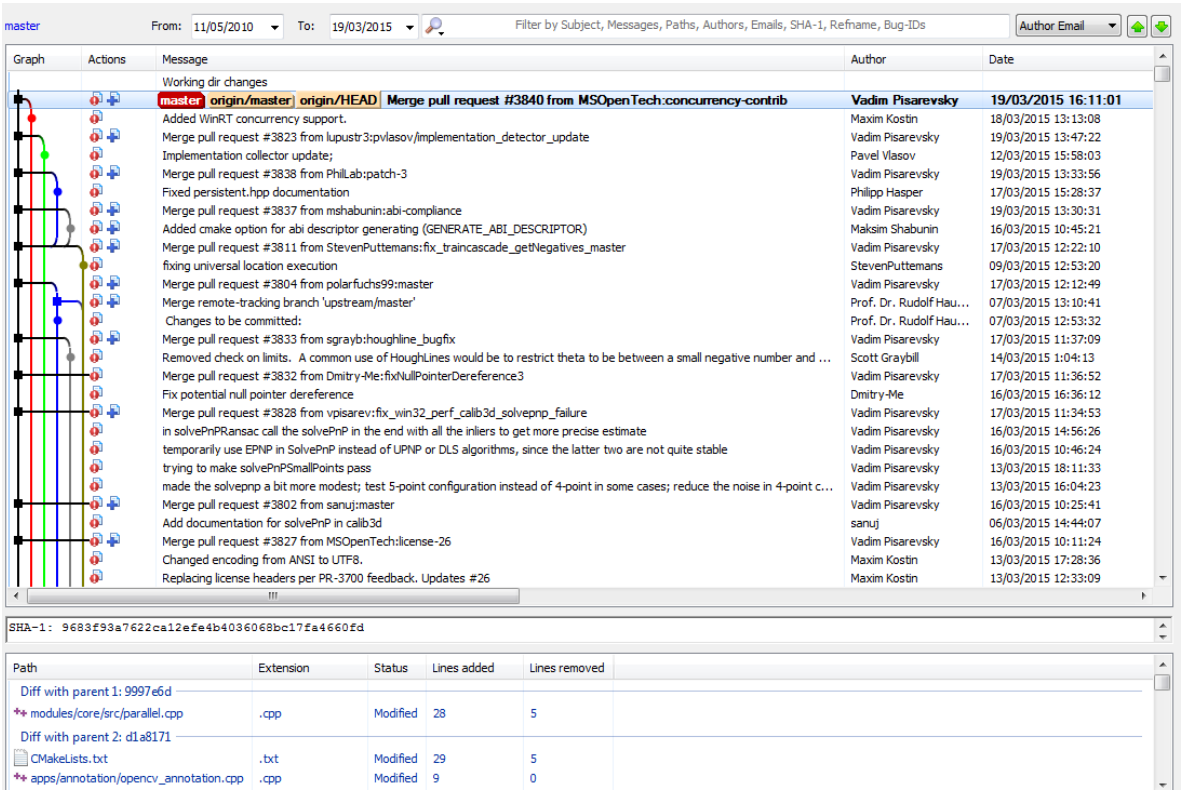


Figura 3. 4:Registro de actividad en el repositorio de OpenCV

Utilizando esta herramienta es muy sencillo comprobar rápidamente quién es el autor del cambio, los archivos modificados o añadidos, y los comentarios al respecto.

Además, tenemos a nuestra disposición múltiples herramientas como por ejemplo el comparador de archivos, que nos permite ver de un simple vistazo los cambios realizados en una archivo de una versión a otra *(Figura 3.5)*.

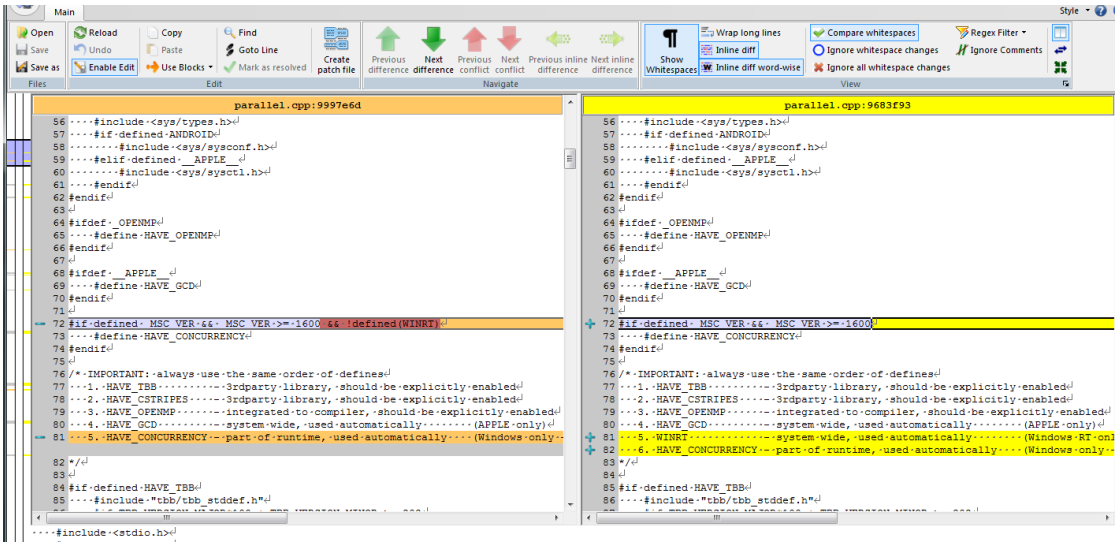


Figura 3. 5: Comparador de archivos (izquierda versión anterior, derecha versión actual)

3.2.1.2 Tortoise Git

Tortoise GIT es una interfaz Shell para GIT , permitiéndonos usar éste de manera gráfica y simple.

3.2.3 CMake

Una vez que tenemos el código fuente es necesario adaptarlo al entorno con el que vamos a trabajar, en este caso Microsoft Visual Studio 2013. Para ello es necesario construir un proyecto con el formato adecuado para que MVS2013 pueda trabajar con él. Utilizando CMake podemos seleccionar el código fuente que hemos clonado desde el repositorio.

Para ello seleccionaremos el directorio donde hemos descargado el código fuente e indicaremos dónde construir la solución *(Figura 3.6)*.

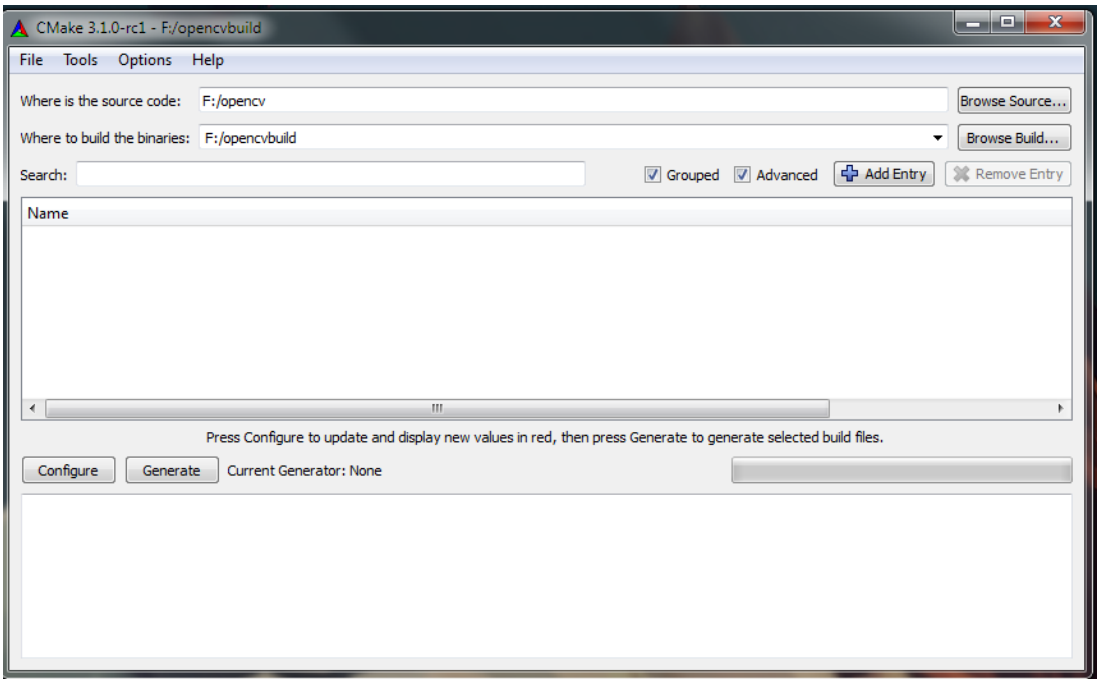


Figura 3. 6: Ventana principal de CMake

Hacemos click en "Configure" y seleccionamos como plataforma MVS2013 *(Figura 3.7)*.

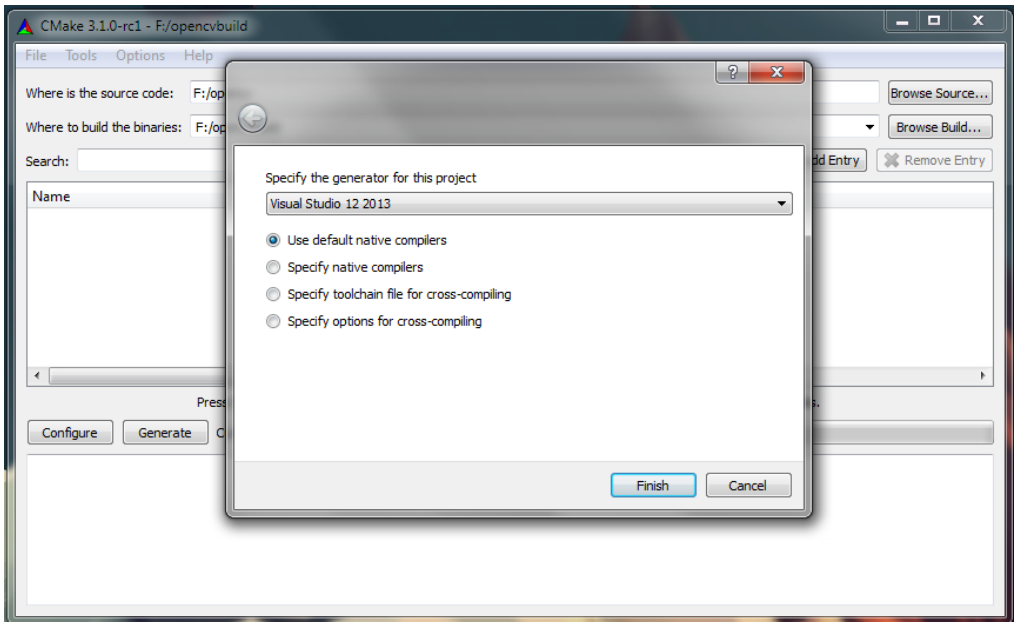


Figura 3. 7: Selección del entorno de trabajo a utilizar

Esperamos a que se detecten los parámetros necesarios para la instalación y obtendremos una salida como la que vemos en la imagen x. En ese punto podremos configurar las características que tendrá nuestra versión de OpenCV, como por ejemplo si queremos construir los ejemplos, la documentación u opciones más avanzadas como soporte para Cuda, IPP, etc. *(Figura 3.8)*.

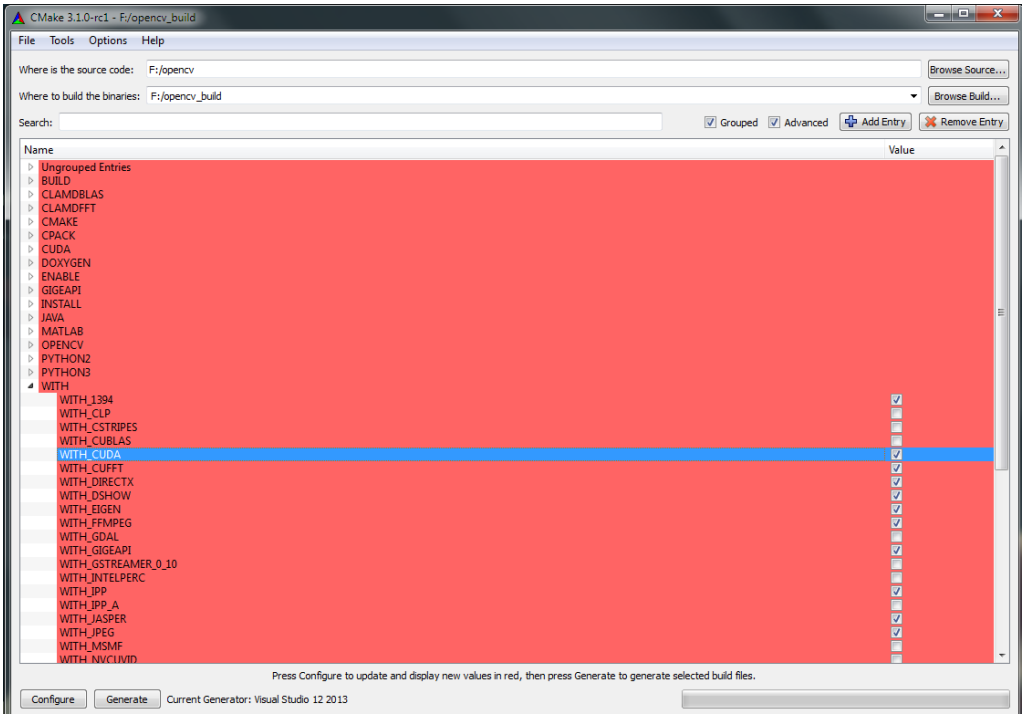


Figura 3. 8: Selección de las características con las que construir OpenCV

Cuando hemos elegido los elementos que se construirán, haremos click en "Generate" y ya tendremos disponible la solución *(Figura 3.9)*.

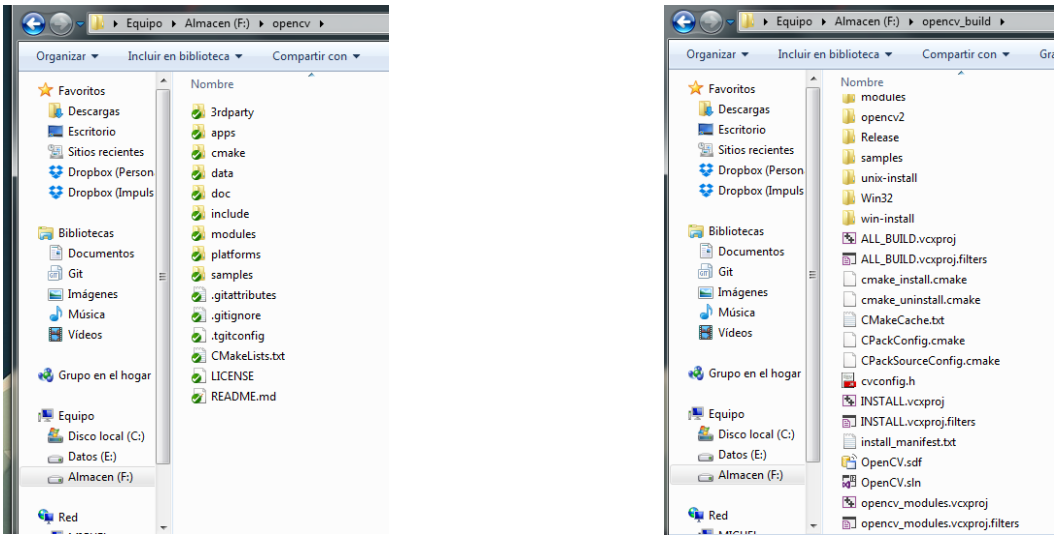


Figura 3. 9: Comparación del código fuente de OpenCV con su solución para Microsoft Visual Studio 2013

3.2.4 Compilación e instalación

Ahora necesitamos compilar OpenCV, para ello abriremos el archivo "Opencv.sln" en la carpeta de la solución y haciendo click derecho sobre el proyecto que nos interese seleccionamos la opción "build".

En este caso seleccionamos "Install", hay que tener en cuenta que nos va interesar utilizar la versión de "Debug" como de "Release" así que se compilarán ambas.

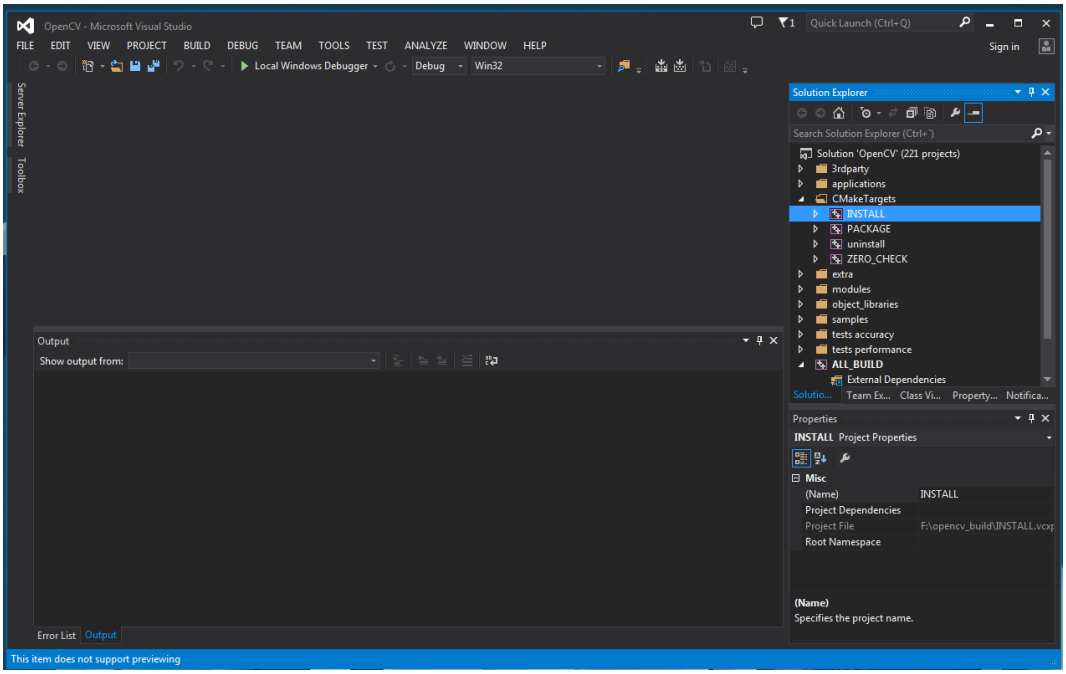


Figura 3. 10: Solución de OpenCV en Microsoft Visual Studio 2013

A partir de este momento ya tendremos las librerías necesarias con las que cualquier proyecto que lo necesite puede trabajar *(Figura 3.10)*.

3.3 Creación de proyectos de OpenCV para Microsoft Visual Studio 2013

La manera más sencilla de crear proyectos es usando CMake ya que permite crear proyectos configurados previamente, es decir, con los links a todas las librerías necesarias, definiciones de pre compilador, etc.

Esto reduce mucho el componente de error humano y permite utilizar un procedimiento "estándar" que no varía independientemente del número de proyectos que creemos.

3.3.1 Creación de proyectos mediante CMake

Empezamos escribiendo el archivo CMakeLists.txt, este archivo es el que posteriormente leerá CMake y construirá el proyecto basándose en lo que se haya especificado en él.

Dentro de este archivo CMakeLists.txt se establece una lista de operaciones a seguir, en este caso se procede de la siguiente manera:

1. Creación de variables indicando el nombre que tendrá el proyecto y el nombre que tendrá el archivo del código fuente.
2. Confirmar que OpenCV se encuentra instalado en la máquina, si no es el caso se cancelará la construcción del proyecto
3. Incluir y linkear el proyecto a los directorios de OpenCV
4. Definir la ruta dónde se creará el ejecutable del proyecto
5. Linkear las librerías de OpenCV
6. Copiar las DLLS de OpenCV necesarias a la carpeta del ejecutable
7. Escribir un archivo con los valores de todas las variables que se han utilizado en la construcción del proyecto, esto se utiliza como método de control para hacer un seguimientos de los pasos seguidos por CMake en caso de que la construcción tenga algún error.

Una vez terminado el CMakeLists.txt ejecutaremos CMake, seleccionando la ubicación del archivo y la ubicación donde se quiere situar la solución para MVS13 tal y como se explica en el punto 2.2.3 de este escrito y se hace click en "Configure" y "Generate"

En este punto ya está sentada la base para poder empezar a programar utilizando todas las funciones de OpenCV.

El código del archivo CMakeLists.txt se puede encontrar en el CD deltrabajo

4. Cámaras

En este proyecto, las cámaras juegan un papel fundamental ya que son el instrumento principal con el que se recoge la información necesaria para el funcionamiento del sistema. Escoger las cámaras correctas así como una configuración adecuada de las mismas es una parte muy importante de este trabajo y debe ser analizada con detenimiento.

4.1 Elección de las cámaras

Para escoger unas cámaras adecuadas primero es necesario definir qué parámetros de las mismas son más importantes para el sistema.

Como primera aproximación, se hizo una investigación [2][4][5] sobre webcams utilizadas en otros proyectos de visión por computador, lo que redujo la lista de candidatos a 2 cámaras: Play Station Eye y Logitech c920 *(Figura 4.1)*.

Figura 4. 1: Play Station Eye (izquierda) y Logitech C920 (derecha)

Teniendo en cuenta que se va a trabajar con webcams y no con cámaras profesionales, la lista de parámetros que se consideraran van a estar definidos de la siguiente manera:

Resolución: indica cuánto detalle se puede observar en una imagen. Una mayor resolución permite obtener una mayor precisión dentro de cada imagen.

Fotogramas por segundo (FPS): Cantidad de fotogramas que es capaz de tomar la cámara en 1 segundo. A mayor FPS se conseguirá una precisión mayor de la trayectoria que siguen las herramientas en el espacio.

Ratio de aspecto: proporción entre el ancho y el alto de una imagen.

Facilidad de integración: que alguna de las webcams cuente con software específico que facilite su integración en OpenCV o para tareas de visión por computador es un factor importante a la hora de tomar una decisión.

De esta manera se puede decir que en términos absolutos, la cámara más optima sería aquella que proporcionase un mayor número de fotogramas por segundo a una resolución mayor.

Por desgracia el ancho de banda es un factor limitante y ambos conceptos están ligados de tal manera que para conseguir FPS altos es necesario trabajar con menores resoluciones y trabajar con resoluciones altas implica obtener unos FPS bajos.

Además es necesario tener en cuenta el tiempo de procesamiento de las imágenes. No sirve de nada tener una cámara capaz de tomar y transmitir imágenes a una resolución y FPS muy altos si el análisis de una imagen o va a estar acabado antes de que llegue la siguiente.

	Resolución máxima	FPS	Ratio de aspecto	Facilidad de integración
PS Eye	640 x 480	Hasta 120	4:3	Drivers y SDK propietarios
Logitech C920	1920 x 1080	Hasta 60	16:9	Plug and play

Tabla 4. 1: Comparativa de prestaciones entre PS Eye y Logitech C920

La cámara escogida es la **Logitech C920** por los siguientes motivos:

- Un ratio de aspecto panorámico es beneficioso ya que cuadra mejor con las dimensiones del espacio de trabajo. Si por cualquier motivo el banco de trabajo contase con limitaciones en el espacio vertical utilizable, podríamos situar las cámaras a una distancia inferior a la que se necesitaría con un ratio de aspecto de 4:3 y seguir captando el banco entero.

- Cuenta con software específico para controlar los parámetros de 2 webcams por separado, cosa nada común en el campo de cámaras de visión por computador no profesionales.

- Aunque no se puede saber antes de adquirirla, se ha encontrado información que apunta que se pueden obtener y procesar resoluciones de 720p a 30 FPS. [2] para proyectos de visión estéreo

4.2 Logitech C920

A continuación se expondrán las características técnicas de la cámara escogida:

Característica	Valor
*Campo de visión Horizontal	70,42º
*Campo de visión Vertical	43.30º
*Campo de visión Diagonal	78º
*Ratio de aspecto del sensor	16:9
Resolución	1920x1080 @30 FPS ** 1280x720 @60 FPS
Ancho de banda requerido	2 Mbps @ 1080p 1 Mbps @ 720p
Compresión	H 264
Peso	227g
Dimensiones	23 x 19,2 x 7,4 mm

Tabla 4. 2: Especificaciones completas Logitech C920

*Dado que no es una cámara profesional, los datos referentes a los campos de visión y tamaño de sensor son cálculos realizados por usuarios de la misma [3] *(Figura 4.2)*.

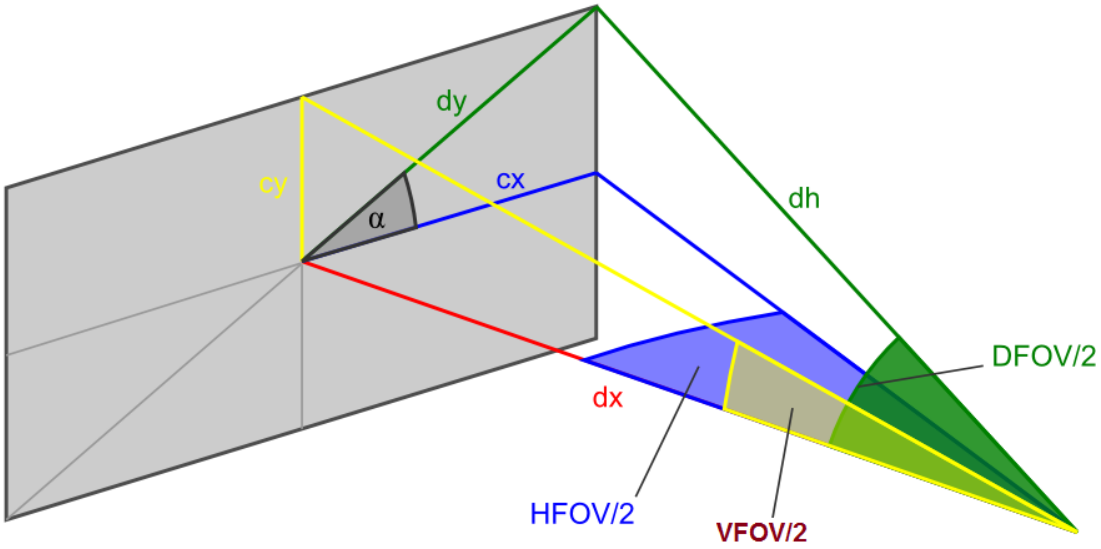


Figura 4. 2: Representación de los campos de visión Horizontal (HFOV), Vertical (VFOV) y Diagonal (DFOV) [3]

** Para alcanzar estos parámetros es necesario utilizar una compresión h.264 que se lleva a cabo dentro de la misma webcam. En líneas generales el algoritmo consiste en crear bloques de 16x16 pixeles y luego a la hora de descodificar se ejecutan 9 tandas de algoritmos (educated guesses) que permiten reconstruir con mayor o menor fidelidad los pixeles originales *(Figura 4.3)*.

Figura 4. 3: Ejemplo de bloques en h.264

4.2.1 Software de configuración

Como se ha apuntado anteriormente, uno de los parámetros de mayor peso a la hora de escoger esta cámara frente a otras es que se utiliza de forma habitual en proyectos de visión por computador, con lo cual se cuenta con bastante apoyo por parte de la comunidad de usuarios técnicos.

Esto supone una gran ventaja frente a otras opciones ya que las posibilidades de detectar y solucionar cualquier problema eventual que pueda surgir son mucho más altas que con otras opciones.

Uno de estos problemas es que normalmente las cámaras de consumo no están preparadas para operar por parejas, de tal forma que no es posible controlar y almacenar los parámetros de configuración de cada cámara por separado (foco, iluminación, etc.).

Sin embargo , existe un programa creado por un usuario de Logitech C920 que permite configurar ambas cámaras de manera independiente, lo que es crucial para aumentar al máximo la repetitividad del sistema *(Figura 4.4)*.

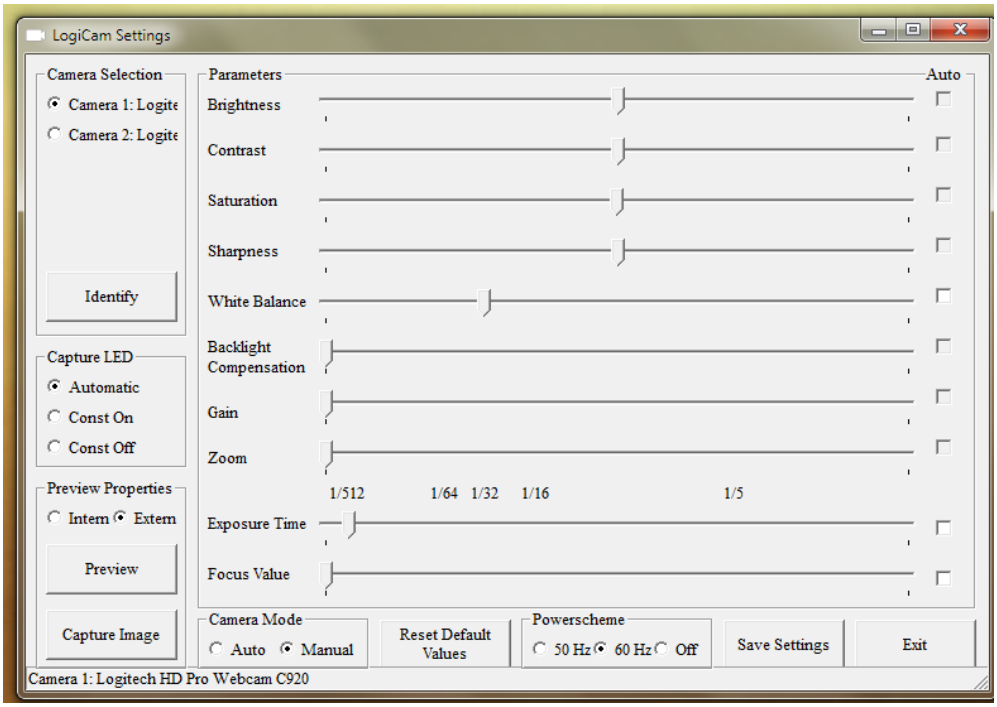


Figura 4. 4: Interfaz del software de configuración de las cámaras.

Gracias a esto se podrán elegir y lo que es más importante, mantener unos parámetros óptimos, eliminando posibles discrepancias y focos de errores.

5. Diseño del espacio de trabajo

En este capítulo se hablará de todo lo relacionado con el aspecto físico del sistema, lo que abarca tanto el propio espacio de trabajo como otros factores como la distribución de las cámaras y el diseño de marcadores.

5.1 Banco de trabajo

Dado que se conocen las medidas y características del banco real donde se pretende utilizar el sistema, se lleva a cabo una reproducción lo más fidedigna posible del mismo para ajustar en la medida de lo posible este proyecto su aplicación real *(Figura 5.1)*.

La construcción de este espacio de trabajo supone un sobrecoste tanto en tiempo como en materiales, sin embargo las ventajas obtenidas son de mucha mayor importancia:

Parámetros constantes: trabajar siempre con el mismo nivel y calidad de luz así como una colocación y orientación de las cámaras constante supone a la larga un ahorro de tiempo debido a la minimización de ajustes necesarios entre 2 experimentos separados en el tiempo.

Dimensiones Reales: esto nos va a permitir anticipar cualquier problema que haría el sistema de monitorización inviable en un entorno real, como podría ser la limitación de espacio, posicionamiento y ajuste de las cámaras, etc.

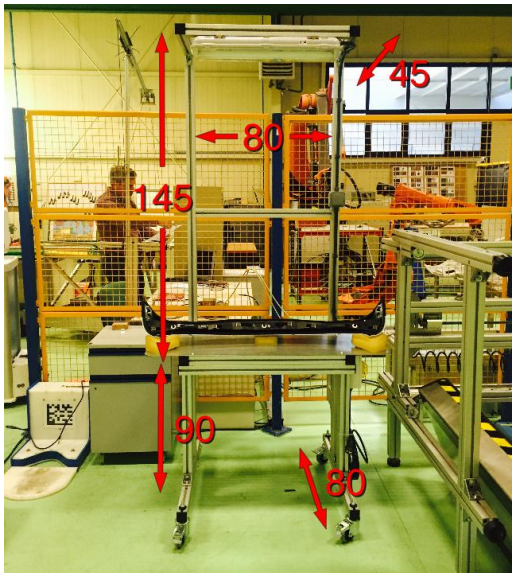


Figura 5. 1: Banco de trabajo real.

La elección de materiales es crucial ya que si se efectúa de manera correcta, tendrá un impacto muy positivo tanto en el aspecto constructivo (menor tiempo de construcción) como de cara a facilitar el tratamiento de imágenes (escogiendo colores y acabados adecuados).

Superficie de trabajo: Tal vez la decisión más importante reside en el color y el acabado de la superficie donde se desarrolla la operación. Dado que no hay requerimientos que especifiquen lo contrario, se utiliza un tablero de color negro mate *(Figura 5.3)*.

La elección del acabado mate se debe a que cuando la luz incide en la interfase entre dos medios, la componente reflejada en cada dirección del espacio es la suma de la componente difusa (zonas mates) y la componente especular (brillos).

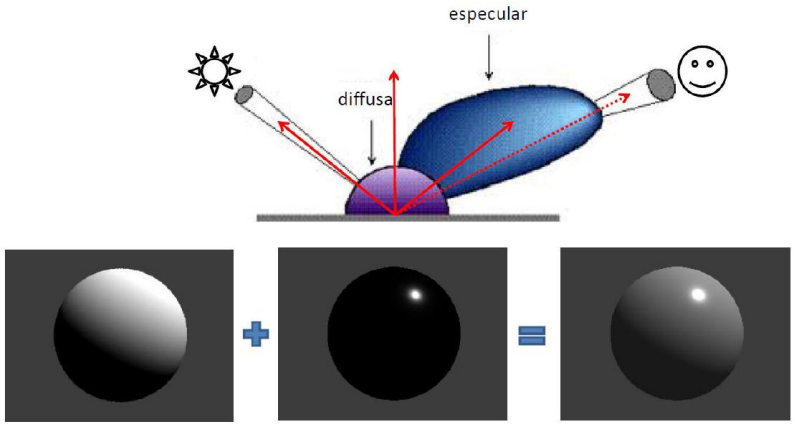


Figura 5. 2: Tipos de reflexión en objetos iluminados.

La componente especular se concentra en la dirección simétrica de la incidencia mientras que la componente difusa es uniforme en el semiespacio. No depende de la dirección de observación y es proporcional a la cantidad de luz que incide en el objeto *(Figura 5.2)*.

El objetivo es minimizar esa componente especular que puede producir saturación en la imagen y dificultar el procesamiento de esta.

Figura 5. 3: Superficie elegida para el banco de trabajo.

Estructura: La elección de la estructura afectará sobre todo al precio final y al tiempo de construcción.

Se barajan dos posibilidades, madera y listones de metal para estanterías prefabricadas.

La madera ofrece una solución económica para la construcción de la estructura, sin embargo requiere de demasiado tiempo de montaje y ajuste en comparación con la segunda opción propuesta.

Además en este punto se considera un recurso más valioso el tiempo que el dinero y dado que la diferencia de precio entre ambas soluciones no es elevada, finalmente se opta por los listones de metal *(Figura 5.4)*.

Figura 5. 4: Forma y distribución final del banco de trabajo.

5.2 Distribución de cámaras

Ya conocemos cómo es el espacio de trabajo y además se sabe que vamos a disponer de una distancia de aproximadamente 1 metro y 20 centímetros entre las cámaras y la zona de trabajo del operario.

Como ya se ha visto en el apartado de este trabajo titulado "La línea base", es recomendable que la distancia entre las lentes sea 1/30 veces la distancia mínima a medir. Si se considera ésta distancia mínima como 1 metro, las lentes deben quedar situadas a una distancia de 3.33 centímetros [13].

Aquí surge el primer conflicto ya que en la visión estéreo tradicional, se considera una distribución de las cámaras de tal manera que estén situadas una al lado de otra (estéreo horizontal), sin embargo las características físicas de las cámaras elegidas hacen imposible situarlas de esta manera y cumplir a la vez las especificaciones de diseño *(Figura 5.5)*.

Esto nos va a obligar a trabajar con ellas en una configuración vertical, y aquí es donde se hace notar el siguiente problema: las cámaras tienen una pinza que hace imposible su colocación una encima de la otra por lo que se tienen 2 opciones:

1. Desmontar la cámara para quitar el soporte
2. Rotar una de las cámaras

Figura 5. 5: Obstáculo que dificulta el posicionamiento de las cámaras.

La primera opción es parcialmente destructiva y dado que se quieren utilizar las cámaras para otros proyectos futuros, se opta por rotar una de las cámaras *(Figura 5.6)*.

Esto supone que a la hora de capturar las imágenes será necesario invertir esa rotación mediante software, pero eso es asumible ya que el coste de recursos de esa operación es mínimo. Los detalles de esta rotación mediante software se pueden ver en la apartado "Captura de imágenes" de este trabajo.

Figura 5. 6: Distribución final de las cámaras.

5.3 Elección de marcadores

Para hacer el sistema lo más estándar y replicable posible, se van a utilizar unos marcadores que cualquiera pueda conseguir, es decir, en vez de crear unos marcadores arbitrarios desde cero se ha partido de los que ya se usan de manera frecuente en el sector de la visión por computador.

OpenCv cuenta con una plantilla "oficial" que se puede utilizar con multitud de algoritmos de la librería. Esta plantilla (conocida como OpenCv Chessboard 9x6) se puede encontrar fácilmente en la página oficial.[1]

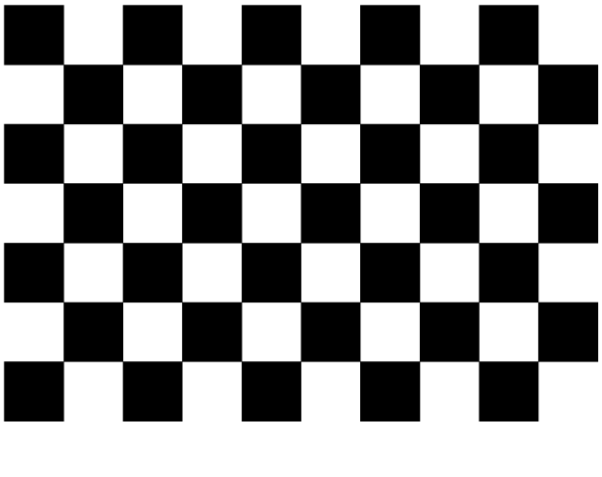


Figura 5. 7: Plantilla "Chessboard" oficial de OpenCV [1].

Con esto ya se han escogido las características físicas de los marcadores, ahora falta es necesario concretar los detalles así que se van a responder estas preguntas: ¿Son necesarios tantos cuadrados? ¿Se puede reducir el tamaño de la plantilla?

Para responder a la primera pregunta se recurre a la documentación de OpenCv. Tras una breve investigación es fácil darse cuenta que para trabajar con estas plantillas siempre se hace uso de la función *findChessboardCorners*(), la cual recibe una imagen en escala de grises y la escanea en busca de una plantilla de cuadrados.

Dicha función tiene como parámetros de entrada (entre otros) el ancho y el largo (en número de cuadrados) de la plantilla que se quiere buscar. Ahondando un poco más en la documentación se descubre que el tamaño mínimo que acepta esa función es de 4x3 cuadrados, así que con eso queda definido el número de cuadrados que tendrán los marcadores.

5.4 Iluminación

Por las características del banco de trabajo real, la iluminación se efectuará con una luz fluorescente situada en lo alto del banco, siendo esto el estándar para este tipo de puestos por eso es la que se va a utilizar en el sistema. Sin embargo considero que no es la mejor opción ya que se van a originar sombras que pueden entorpecer el funcionamiento del sistema.

El uso de una iluminación difusa que alcance el banco de trabajo desde multitud de ángulos sería lo ideal en este caso, pero dadas las características físicas del mismo su implementación no es posible ya que podría entorpecer el trabajo del operario a parte de encarecer conjunto del sistema debido a la necesidad de instalar un sistema de iluminación en cada puesto.

6. Diseño del software

En éste capítulo se expone detalladamente la manera en la que opera tanto el software principal como el resto de programas asociados así como la manera en que estos se relacionan. Con ello se pretende dejar clara tanto la estructura del sistema como la manera en que trabaja cada parte .

6.1 Visión general

Por motivos de claridad y facilidad de implementación, se han decidido separar ciertas características que si bien son imprescindibles para el correcto funcionamiento del sistema, no forman en sí mismas parte de él.

Dentro de estas características podemos encontrar los apartados de calibración y configuración del programa en general.

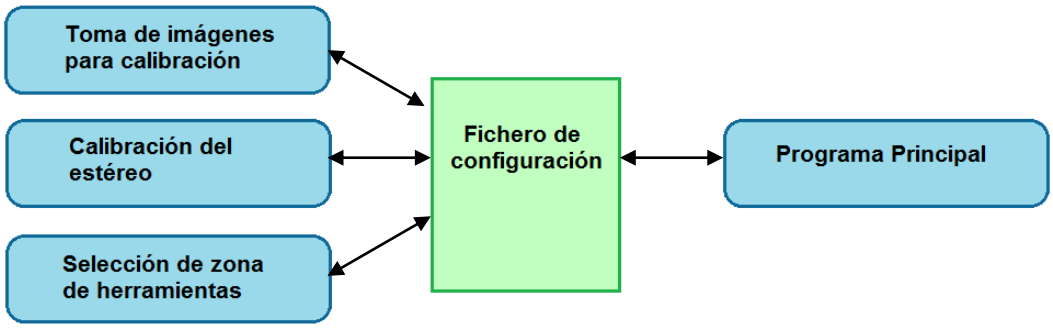


Figura 6. 1: Relación entre los diferentes programas involucrados.

6.2 Fichero de configuración

Dado que para el funcionamiento de este sistema es necesario el uso de varios programas y los parámetros necesarios para su ejecución, aunque configurables, deben ser consistentes para todos, se ha decidido implementar un archivo de configuración.

Este archivo es único y será leído al principio de la ejecución de cada programa, garantizando así que se utiliza en todos los mismos parámetros y acotando el gran medida los errores que se pueden producir debido a posibles despistes u olvidos.

El ejemplo más claro es la resolución a utilizar. El usuario debe ser capaz de escoger la que desee pero esta debe ser la misma tanto para el programa de calibración del estéreo como para el programa principal.

Aunque estos parámetros son bastante autoexplicativos, a continuación se detalla para qué sirve cada uno de ellos:

FRAME_WIDTH: Especifica el ancho de la imagen a capturar

FRAME_HEIGHT: Especifica el alto de la imagen a capturar

CALIBRATION_IMAGES_FOLDER_PATH: Ruta donde guardar / leer las imágenes de calibración del sistema estéreo

CALIBRATION_IMAGE_LIST_PATH: Ruta dónde se puede encontrar el archivo con la lista de rutas de las imágenes de calibración

CALIBRATION_IMAGES_NUMBER: Número de imágenes a tomar para la calibración

CALIBRATION_BOARD_HEIGHT: Altura (en número de cuadrados) de la plantilla que se utilizará para la calibración.

CALIBRATION_BOARD_WIDTH: Anchura (en número de cuadrados) de la plantilla que se utilizará para la calibración.

INTRINSICS_FILE_PATH: Ruta donde guardar / leer el archivo de los valores intrínsecos de las cámaras.

EXTRINSICS_FILE_PATH: Ruta donde guardar / leer el archivo de los valores extrínsecos de las cámaras.

TEMPLATE_GRID_HEIGHT: Altura (en número de cuadrados) de la plantilla que se utilizará para el seguimiento.

TEMPLATE_GRID_WIDTH: Anchura (en número de cuadrados) de la plantilla que se utilizará para el seguimiento

STEREO_Scale: Valor del tipo float que especifica si se quiere escalar las imágenes (por defecto 1.0)

HAND_DETECTION_ROI: Ruta de la imagen que se usará de máscara para determinar la zona de herramientas

HAND_DETECTION_EVERY_N_FRAMES: Número entero que especifica cada cuántos fotogramas se efectuará la comprobación de movimiento en la zona de herramientas.

HAND_DETECTION_THRESHOLD: Número entero que determina el umbral para considerar que se ha detectado movimiento en la zona de herramientas.

INLIER_THRESHOLD: Número flotante que especifica el umbral para determinar inliers en la detección A-kaze

AKAZE_THRESHOLD: Número flotante que especifica el umbral de deteción A-kaze

TOOL_DETECTION_THRESHOLD: Número entero que indica el umbral a partir del cuál se considera que se ha detectado una herramienta

TOOL_LIST_FILE: Ruta del archivo que contiene las rutas de todas las plantillas de herramientas para la detección A-Kaze.

FAST_MODE_DETECTION_THRESHOLD: Número entero que indica el umbral del número de pixeles de la máscara a partir del cual se considera un movimiento delicado.

ROTATE: Valor booleano que indica si se quiere trabajar con las imágenes rotadas o sin rotar

DEBUG: Número entero que indica si se quiere recibir feedback de los distintos pasos que se están llevando a cabo

6.3 Programas independientes

Dado que en este capítulo se cubre sólo el tema del diseño del software, a continuación se va a explicar cómo trabaja cada programa así como los parámetros que utiliza para su ejecución. Se puede encontrar una explicación de cómo utilizar estos programas en el apartado "Uso del Software" de este trabajo.

6.3.1 Toma de imágenes para calibración

Este programa se limita a tomar un número de imágenes determinado por el usuario desde el archivo de configuración y guardarlas en la ruta seleccionada también a través de dicho archivo.

Dado que esta captura de imágenes se utilizará para calibrar el sistema estéreo, dicha captura se realiza desde ambas cámaras a la vez obteniendo una lista secuencial.

Se necesitan los siguientes parámetros del archivo de configuración:

FRAME_HEIGH

FRAME_WIDTH

CALIBRATION_IMAGES_FOLDER_PATH

CALIBRATION_IMAGES_NUMBER

ROTATE

6.3.2 Calibración del estéreo

Utilizando las imágenes capturadas previamente, se llevan a cabo todas las operaciones matemáticas necesarias para calcular y almacenar los parámetros intrínsecos y extrínsecos de las cámaras.

En el archivo de configuración será necesario aportar:

CALIBRATION_BOARD_HEIGHT

CALIBRATION_BOARD_WIDTH

CALIBRATION_IMAGE_LIST_PATH

INTRINSICS_FILE_PATH

EXTRINSICS_FILE_PATH

6.3.3 Selección de zona de herramientas

Este programa crea una imagen binaria que representa la zona donde se situarán las herramientas, de tal manera que dicha zona se considere de valor 1 y el resto de 0.

Esta máscara se usará en el bucle principal para determinar si el operario está o no cogiendo una herramienta.

En el archivo de configuración será necesario aportar:

FRAME_HEIGHT

FRAME_WIDTH

HAND_DETECTION_ROI

ROTATE

6.4 Programa principal

La estructura del programa se ha llevado a cabo de manera modular, de tal forma que puede ser fácilmente entendible y modificable ,además de ofrecer una escalabilidad muy buena. Esto es importante ya que facilita mucho el uso de este trabajo como punto de partida para futuros desarrollos *(Figura 6.2)*.

Teniendo esto en cuenta a continuación se expondrá cada "módulo" o función por separado, detallando su funcionamiento y haciendo hincapié en los aspectos más importantes del mismo.

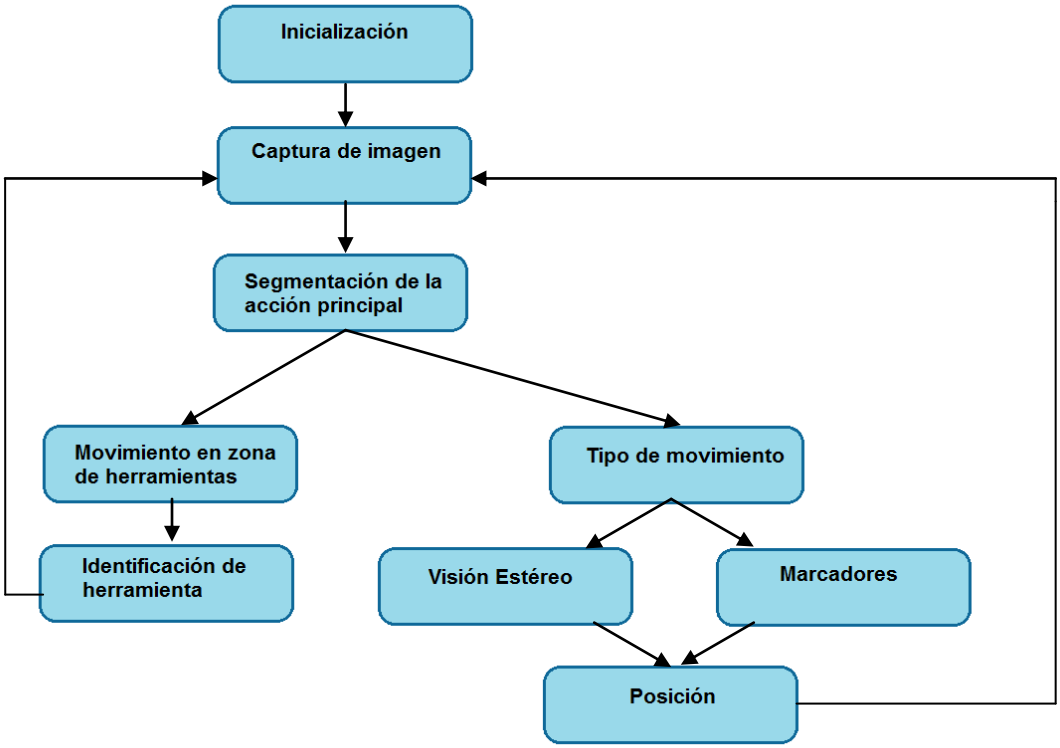


Figura 6. 2: Algoritmo de trabajo usado en el programa principal.

6.4.1 Inicialización

Al ejecutarse el programa se lleva a cabo una inicialización de todos los valores y herramientas necesarios para su funcionamiento, el procedimiento se efectuará siguiendo esta secuencia:

1. **Carga de parámetros de configuración:** Se efectúa una lectura del fichero que contiene todos los parámetros configurables del sistema. Estos parámetros especifican por ejemplo la resolución a utilizar o dónde situar los ficheros de calibración de las cámaras, etc. Para más información a cerca de este fichero consultar el apartado "Archivo de configuración" de este trabajo. Además de leer y cargar todos los valores del fichero de configuración , se cargan también todas las imágenes necesarias como las plantillas para el A-Kaze o la máscara para la zona de herramientas.
2. **Encendido de las cámaras:** se comprueba que efectivamente el sistema tiene acceso a 2 cámaras y se abren, especificando la resolución con la que se va a trabajar.
3. **Configuración de las cámaras:** se abren un par de ventanas donde se pueden visualizar ambas cámaras. En este punto se deben configurar por ejemplo el tiempo de exposición, el foco, etc.

Una vez inicializado el sistema se da paso al bucle principal que se ejecutará continuamente. Dentro de este bucle se irá llamando a las funciones según sea necesario *(Figura 6.3)*.

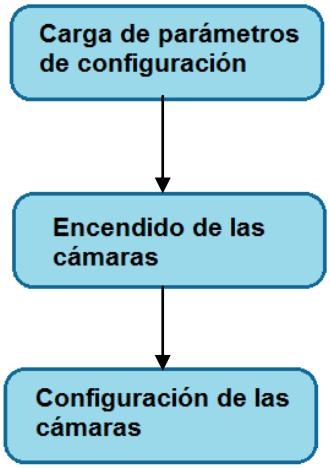


Figura 6. 3: Acciones realizadas al inicio del programa.

6.4.2 Captura de imágenes

Como es lógico, con cada ejecución del bucle principal se llevará a cabo una nueva captura de imágenes. Además interesa tratar esas imágenes de tal forma que puedan ser utilizadas por el resto de funciones ya que muchas requieren por ejemplo el uso de imágenes en escala de grises *(Figura 6.4)*.

Para ello se va a trabajar siguiente protocolo:

1º Captura: toma una imagen a color desde ambas cámaras y la almacena.

2º Tratamiento:

Caso 1:Se está ejecutando el módulo de visión estéreo:

Se rectifican usando los parámetros de calibración

Se transfieren a la memoria de la tarjeta gráfica

Caso 2: No se está ejecutando el módulo de visión estéreo

Se convierten las imágenes a escala de grises

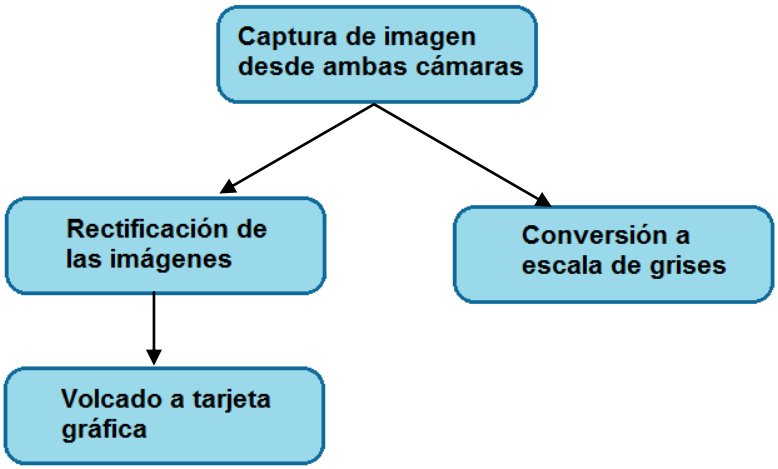


Figura 6. 4: Procedimiento de captura de imágenes.

6.4.3 Segmentación de la acción principal

Inmediatamente después de adquirir una imagen e independientemente del módulo que esté en ejecución, se llevará a cabo una segmentación con el objetivo de obtener la zona donde se está llevando a cabo la acción principal *(Figura 6.5)*.

Este paso se efectuará siempre ya que su coste computacional es mínimo hace que todas las operaciones posteriores que se realicen sean mucho más rápidas.

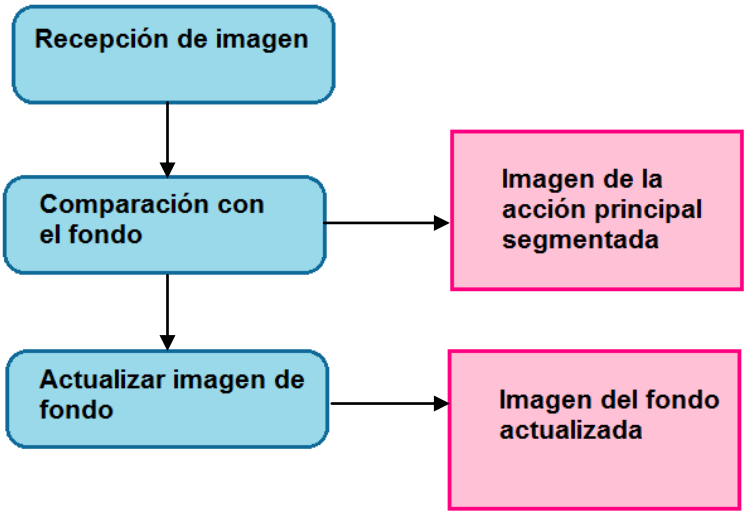


Figura 6. 5: Procedimiento de segmentación de la acción principal.

6.4.4 Identificación del cambio de herramientas

Para identificar el cambio de herramientas se va a suponer que el operario tiene un área conocida donde están situadas las mismas, de esta forma cuando se detecte movimiento en dicho área, se podrán llevar a cabo las operaciones pertinentes.

Esta comprobación se llevará a cabo comparando la máscara de la acción principal obtenida en el paso previo con otra máscara definida previamente por el usuario que indique la zona de herramientas (ver apartado "Selección de la zona de herramientas" de este trabajo).

Si el número de píxeles resultantes de la intersección de esas 2 máscaras supera cierto umbral, se considera que el operario ha entrado en la zona y el programa salta a la detección de herramientas *(Figura 6.6)*.

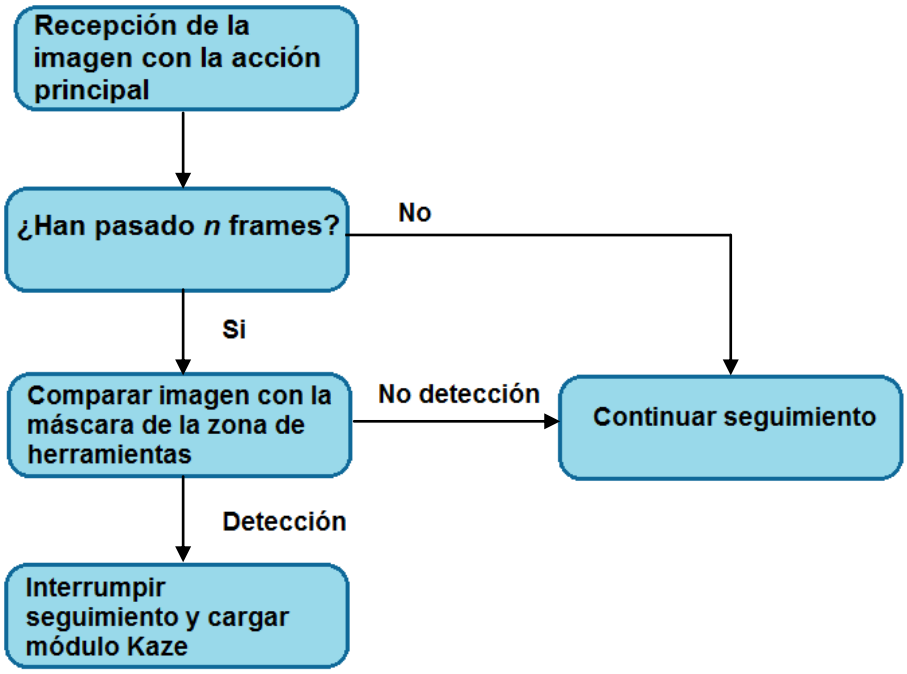


Figura 6. 6: Procedimiento de identificación del cambio de herramientas.

Como el cambio de herramientas no es una operación frecuente en comparación con el número de fotogramas que se analizan, se establece que sólo se compruebe que el operario entra en la zona de herramientas cada n fotogramas para minimizar el coste computacional.

6.4 5 Identificación de la herramienta en uso

Cuando se haya detectado que el operario se encuentra situado dentro de la zona de herramientas, las imágenes pasan a ser procesadas por la función de detección A-Kaze.

Dentro de esta función se comparará la imagen actual con todas las plantillas de herramientas de las que se dispone. Cuando se encuentra una plantilla con un número de coincidencias mayor que cierto umbral especificado, se considera una detección correcta y se deja paso al seguimiento de operaciones *(Figura 6.7)*.

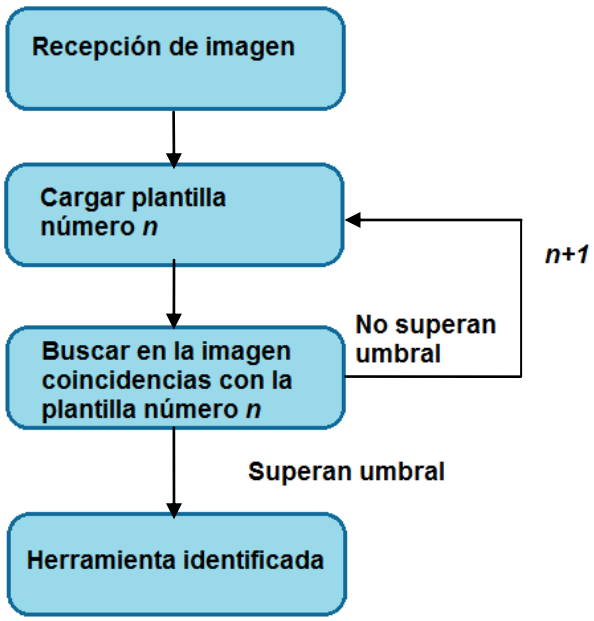


Figura 6. 7: Procedimiento de identificación de herramientas en uso.

6.5 Seguimiento de operaciones

El seguimiento de operaciones se va a llevar a cabo de dos maneras diferentes, esto es debido a que, como se verá a continuación, ambas tienen sus ventajas y sus inconvenientes pero se complementan y se pueden coordinar para conseguir un seguimiento fluido.

Se parte de la premisa de que existen dos tipos de movimientos, aquellos relevantes para el correcto desarrollo de las operaciones y los que son consecuencia de la necesidad de moverse entre un sitio de operación y otro.

La diferenciación se llevará a cabo suponiendo que por lo general los movimientos que requieren más precisión son más lentos que aquellos que no son tan relevantes.

Para llevar a cabo la diferenciación nos vamos a aprovechar de la propiedad ya explicada del algoritmo de segmentación según la cual, si un objeto de la acción principal pasa el tiempo suficiente en el mismo sitio, se empezará a considerar como parte del fondo.

Sabiendo esto, se pueden contar los pixeles de valor no nulo en la imagen resultante de la detección de movimiento y contar el número de pixeles con valor distinto de 0 en la misma. Si el número de pixeles cae por debajo de cierto umbral, se considera que el movimiento es delicado y se pasa a utilizar la estimación de posición por plantilla *(Figura 6.8)*.

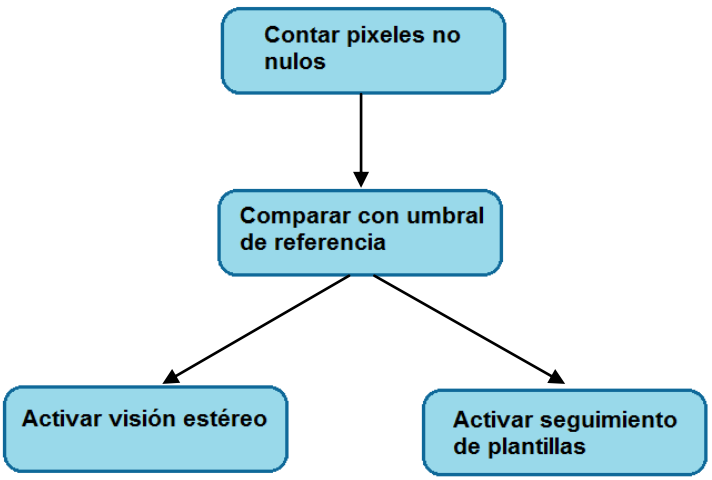


Figura 6. 8: Procedimiento para el seguimiento de operaciones.

Cuando la estimación de la posición se lleve a cabo utilizando la plantilla, obtendremos directamente tanto sus coordenadas dentro de la imagen (x, y) como su orientación en el espacio.

En el modo estéreo nos interesa saber en qué región del espacio ocupa todo elemento en movimiento de la escena (conjunto brazos- herramienta), ósea que interesa el mapa de profundidades al completo, el cual se convertirá en una nube de puntos y se hallará la recta que mejor se ajuste a dicha nube, utilizando los parámetros que la definen como los elementos a controlar.

Esto se tratará más en profundidad en el siguiente apartado, de momento nos interesa solo tener una visión general de cómo interactúan ambas detecciones y cuáles serán los datos obtenidos al final de toda esta operación *(Figura 6.9)*.

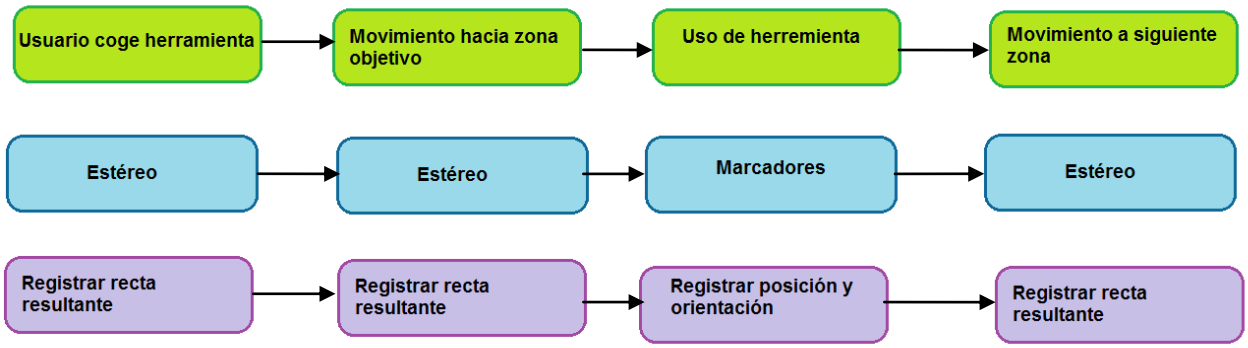


Figura 6. 9: Ejemplo de los diferentes tratamientos y datos recogidos en función de la acción que se esté llevando a cabo.

6.5.1 Estimación de posición estéreo

El interés de esta operación consiste en obtener información tanto de los brazos del usuario como de la herramienta, por lo tanto se utiliza el mapa de profundidades de la escena segmentado con la acción principal.

Haciendo esto, se puede relacionar la posición y el color de los pixeles del mapa de profundidades con una nube de puntos de la siguiente manera:

$$Imagen\ (x) = \ Plano\ (x)$$

$$Imagen\ (y) = \ Plano\ (y)$$

$$Valor\ Pixel\ (Imagen\ (x, y)) = \ Plano\ (z)$$

Una vez se obtienen las coordenadas de la nube de puntos es posible efectuar una regresión lineal que nos de como resultado una recta cuya pendiente y posición representa de manera única la acción que se ve en escena*(Figura 6.10)*.

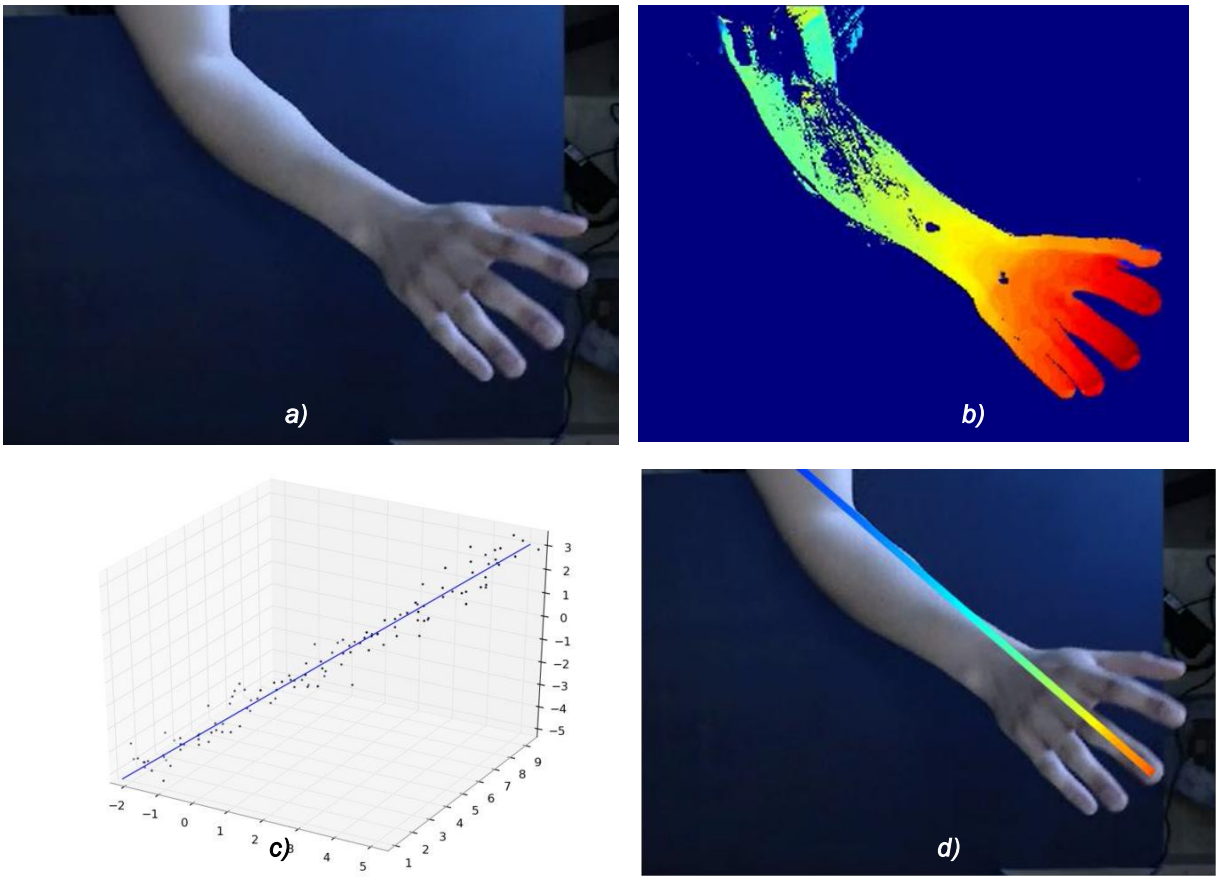


Figura 6. 10: a) Imagen capturada. b) Mapa de profundidades. c) Recta calculada. d) Recta superpuesta cuya pendiente se indica mediante color y superpuesta a la imagen.

Una vez se tiene la recta, se procede a coger dos puntos que pertenezcan a ella para, mediante trigonometría, hallar el ángulo de la recta con respecto al plano Z y respecto al plano X así como el punto en el que corta al plano Z.

Se han escogido estos valores como dato de salida del algoritmo ya que no solo definen completamente la recta obtenida si no que permiten comparar unas con otras de manera sencilla e intuitiva.

Para explicar esto con un ejemplo vamos a suponer que tenemos una recta que pasa por los puntos A=(0,0,0) y B=(3,3,3) *(Figura 6.11)*.

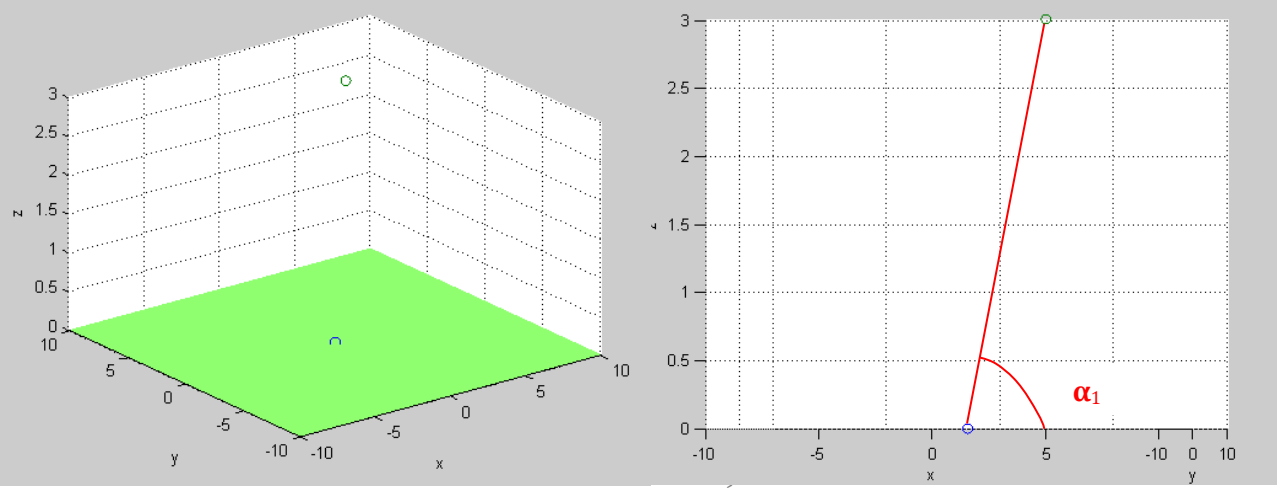


Figura 6. 11: Plano Z=0 (Verde) , puntos A=(0,0,0), B=(3,3,3). y Ángulo con el plano X de la recta que une A y B.

Si se quiere saber el ángulo que tiene esa recta con respecto al plano Z (recordemos que Z=0 equivale a la mesa del banco de trabajo) simplemente se proyectan los puntos en dicho plano, obteniendo así α_1, si se realiza la misma operación pero esta vez proyectando los puntos en el plano Y se obtiene β_1 *(Figura 6.12)*.

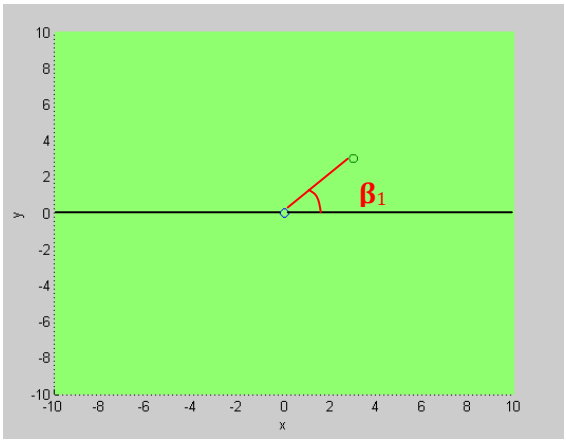


Figura 6. 12: Ángulo con el plano Y de la recta que une A y B.

De esta manera, el resultado final de la detección estéreo son los 3 valores que representan recta que mejor se adapta al conjunto brazo herramienta:

$$\alpha_1 = 45º$$

$$\beta_1 = 45º$$

Punto de corte con el plano Z=(0,0,0)

Estos datos se pueden utilizar posteriormente para determinar qué gesto aparece en pantalla y mantener un registro de los mismos.Cada gesto u operación cuenta con una recta distintiva que permite diferenciarlos en cierta medida de los demás *(Figura 6.13)*.

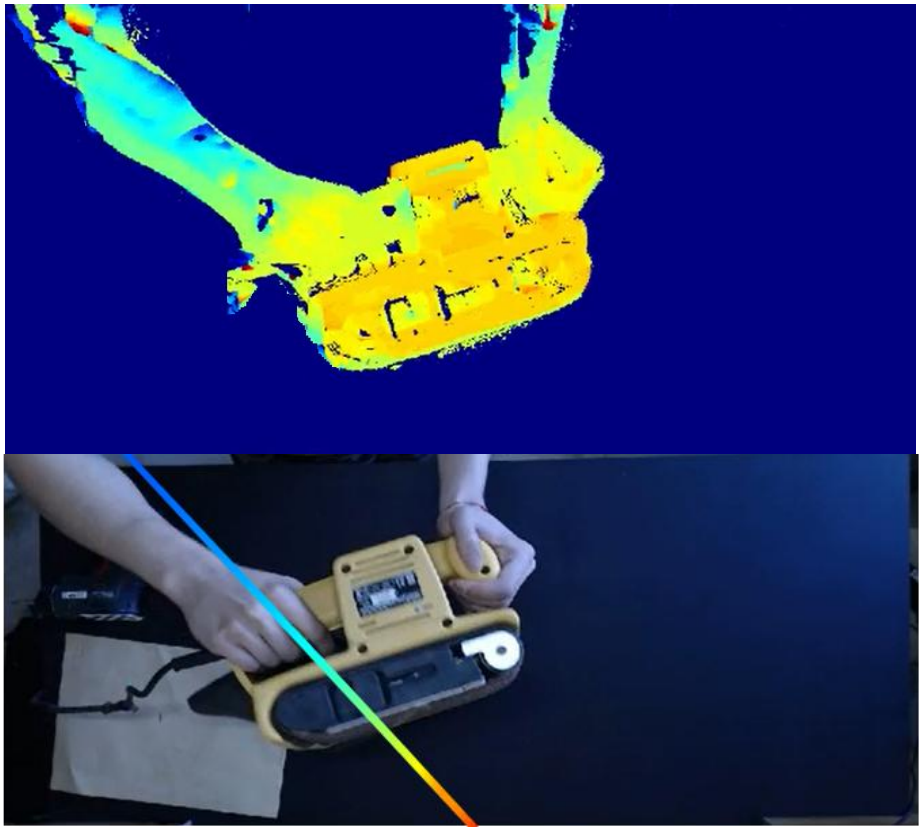


Figura 6. 13: (Parte Superior) Mapa de profundidades de la acción. (Parte Inferior) Recta correspondiente a la acción realizada.

6.5.2 Estimación de posición plantillas

En este caso los datos que se obtienen son los aportados por el algoritmo presente en OpenCV: coordenadas espaciales definiendo la posición de la plantilla e información relativa la orientación de la misma *(Figuras 6.14 y 6.15)*.

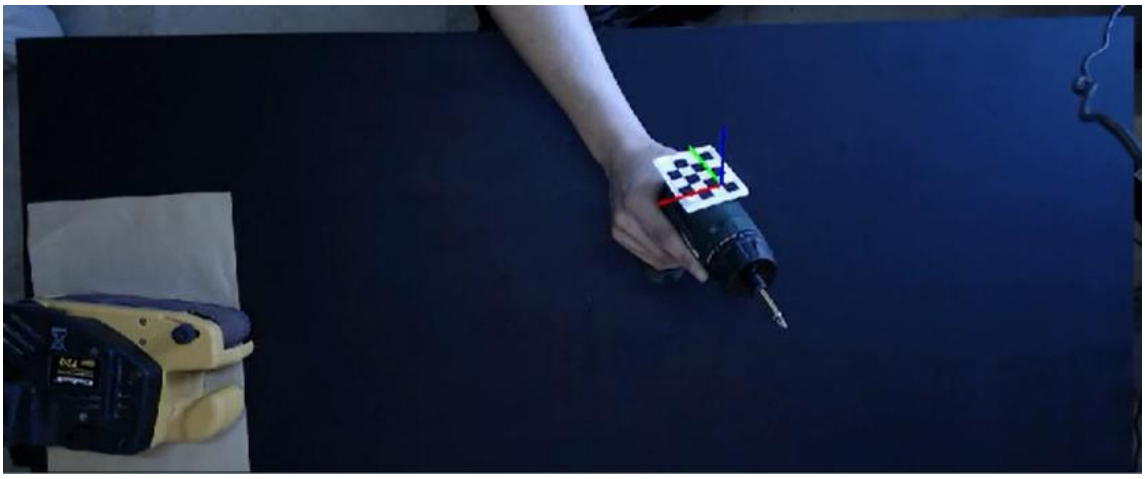


Figura 6. 14: Sistema de coordenadas obtenido de la detección por plantillas.

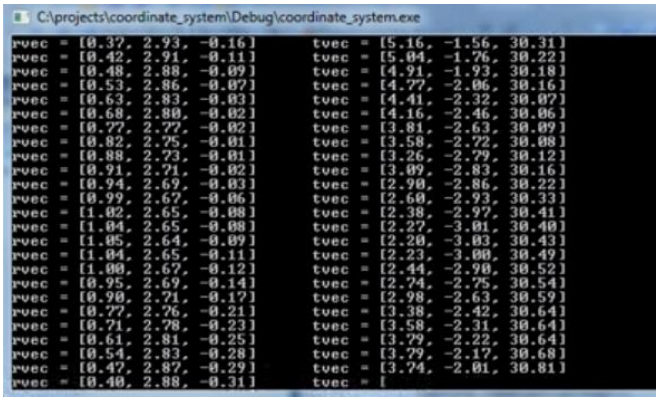


Figura 6. 15: Datos de salida de la detección por plantillas. Dichos datos incluyen el centro de la detección y la orientación de la misma.

Como en el caso anterior, estos datos son registrados y guardados para su posterior estudio.

7. Experimentación y resultados

7.1 Desarrollo enfocado a CUDA o GPU

Al principio de este trabajo, se empezó desarrollando el sistema utilizando la versión clásica de OpenCv que utiliza una implementación basada en CPU en sus algoritmos.

Hay que destacar que esta implementación utiliza tanto Integrated Performance Primitives (IPP) como Thread Building Blocks (TBB) (ambos se explican más a delante) para optimizar los tiempos de ejecución de las operaciones, por lo que lo normal es que con esta versión se consigan unos buenos tiempos de ejecución en general.

Thread Building Blocks (TBB): biblioteca basada desarrollada por Intel para facilitar la escritura de programas que exploten las capacidades de paralelismo de los procesadores con arquitectura multinúcleo.

Proporciona algoritmos y estructuras de datos que permiten al programador evitar en parte las complicaciones derivadas del uso de los paquetes nativos de gestión de hilos de ejecución en los que la creación, sincronización y destrucción de los hilos es explícita y dependiente del sistema. En lugar de esto, la biblioteca abstrae el acceso a los múltiples procesadores permitiendo que las operaciones sean tratadas como tareas que se reparten automática y dinámicamente entre los procesadores disponibles mediante un gestor en tiempo de ejecución *(Figura 7.1)*.

Esta aproximación hace que Intel TBB se incluya en la familia de soluciones para la programación paralela que permiten desacoplar la programación de las características particulares de la máquina.

```
int main(int argc, char* argv[])
{
    cv::Mat img, out;
    img = cv::imread(argv[1]);
    out = cv::Mat::zeros(img.size(), CV_8UC3);

    // create 8 threads and use TBB
    cv::parallel_for_(cv::Range(0, 8), Parallel_process(img, out);
    return(1);
}
```

Figura 7. 1: Ejemplo de uso del Pararell For.

Integrated Performance Primitives (IPP): ofrece rutinas de bajo nivel con muy alto rendimiento, dichas funciones están diseñadas para maximizar el rendimiento de las operaciones en procesadores de la marca Intel y a diferencia de TBB no se pueden utilizar con otra arquitectura.

Todas las funciones de OpenCv están preparadas para utilizar IPP en caso de que sea posible, obteniendo unos rendimientos mucho mayores que cuando se están ejecutando las implementaciones estándar.

Sabiendo esto se realizan unas pruebas de rendimiento para el cálculo de mapas de profundidad utilizando como variable el número de disparidades que utiliza la función estéreo *(Figura 7.2)*.

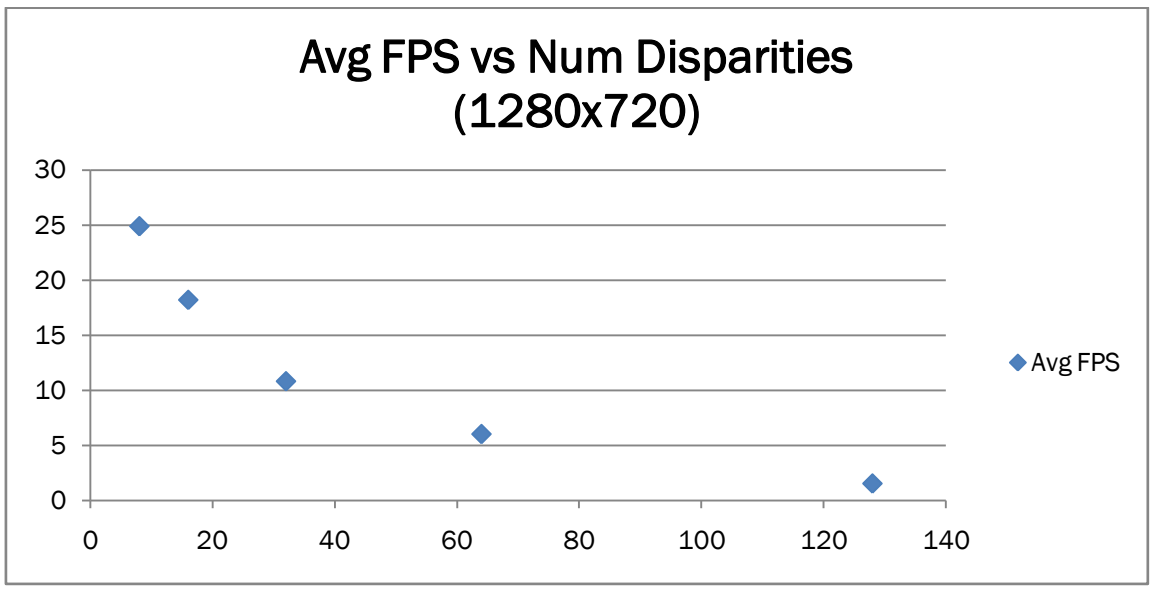


Figura 7. 2: Representación de la variación en los FPS obtenidos según el número de disparidades utilizando la CPU para el cálculo.

Por pruebas anteriores sabemos que el número de disparidades debe ser mayor que 32 para obtener un buen mapa de profundidades. Además los FPS no deberían bajar de 30 para que la toma de datos sea lo suficientemente rápida como para poder seguir los movimientos de la persona.

A raíz de estos resultados se descarta el uso únicamente de la CPU y se empieza a considerar la GPU como elemento de cálculo del mapa de profundidades.

Se repite el mismo experimento pero utilizando la GPU para el cálculo del mapa de profundidades *(Figura 7.3)*.

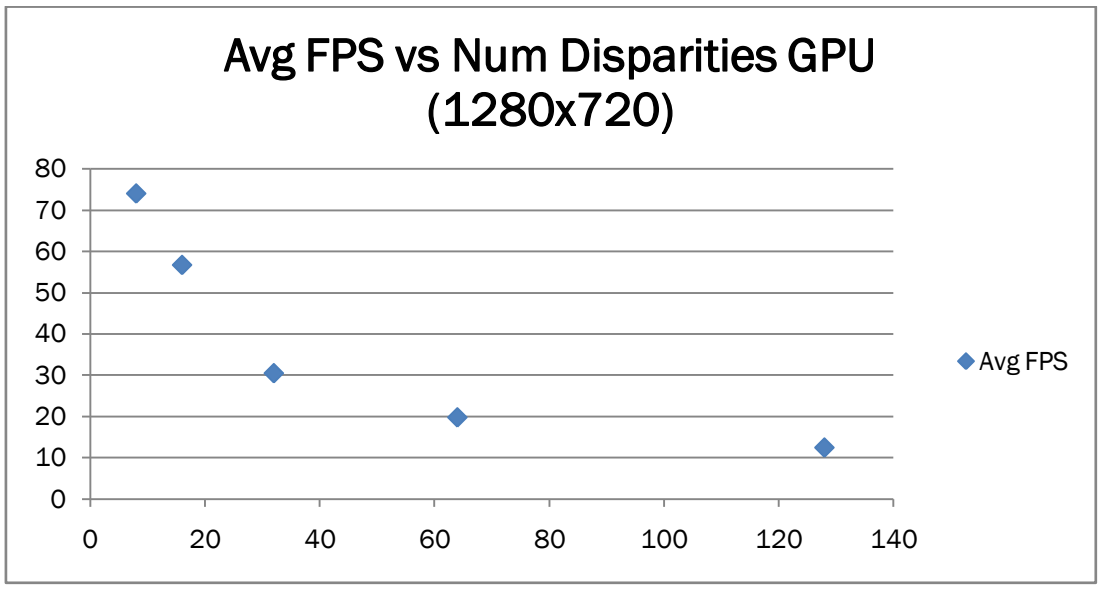


Figura 7. 3: Representación de la variación en los FPS obtenidos según el número de disparidades utilizando la GPU para el cálculo.

Como se puede comprobar, el aumento de rendimiento es notable, permitiéndonos incluso utilizar 64 disparidades para obtener mayor precisión.

A la vista de estos resultados se decide optar por este método asumiendo las concesiones que se explicaron en el apartado "CUDA" tales como que la implementación sólo es válida para tarjetas gráficas de la marca Nvidia.

7.2 Tipo de medición 3D: Estéreo o plantilla

Uno de los objetivos de este trabajo es obtener un sistema que interfiera lo menos posible con el usuario, por eso en un principio se empieza a trabajar utilizando únicamente la visión estéreo aunque se consideran también otras opciones.

7.2.1 Estéreo: Precisión y repetitividad

Vamos a comenzar hablando del sistema estéreo. Para comprobar su rendimiento se ha diseñado un experimento que permita determinar tanto su precisión como su repetitividad.

El objetivo del experimento es simular una acción real minimizando el número de variables no controlables así que interesa es coger una acción lo más sencilla posible. En este caso se simulará que el operario mueve el conjunto brazo-herramienta de manera uniforme y siempre manteniendo el contacto con la mesa *(Figura 7.4)*.

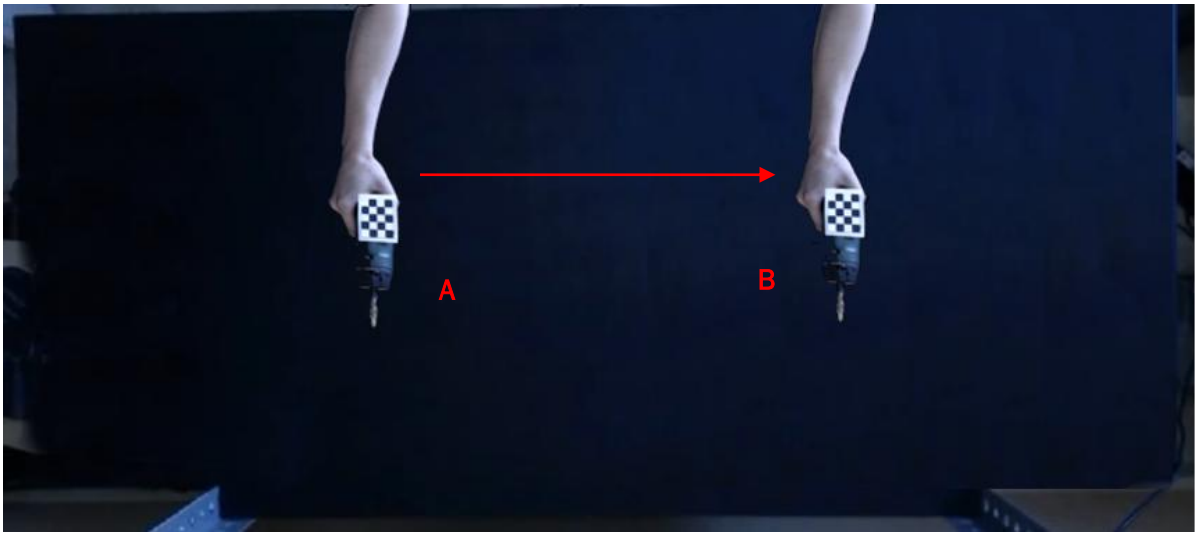


Figura 7. 4: Recorrido de la prueba a realizar.

Además de una acción simple, hace falta que la posición alcanzada al final del movimiento sea siempre la misma, por ello se sustituye el conjunto brazo-herramienta por un instrumento diseñado a tal efecto. y el cual debe mantener unas dimensiones constantes a lo largo del movimiento, permitiéndonos variar su posición dentro del espacio de trabajo pero sin que esto afecte a sus características físicas.

Con esto en mente se crea un sustituye el brazo del operario por un listón de metal y la herramienta por una lata de dimensiones conocidas, obteniendo una solución simple pero efectiva que permite realizar los experimentos bajo unas condiciones controladas *(Figura 7.5)*.

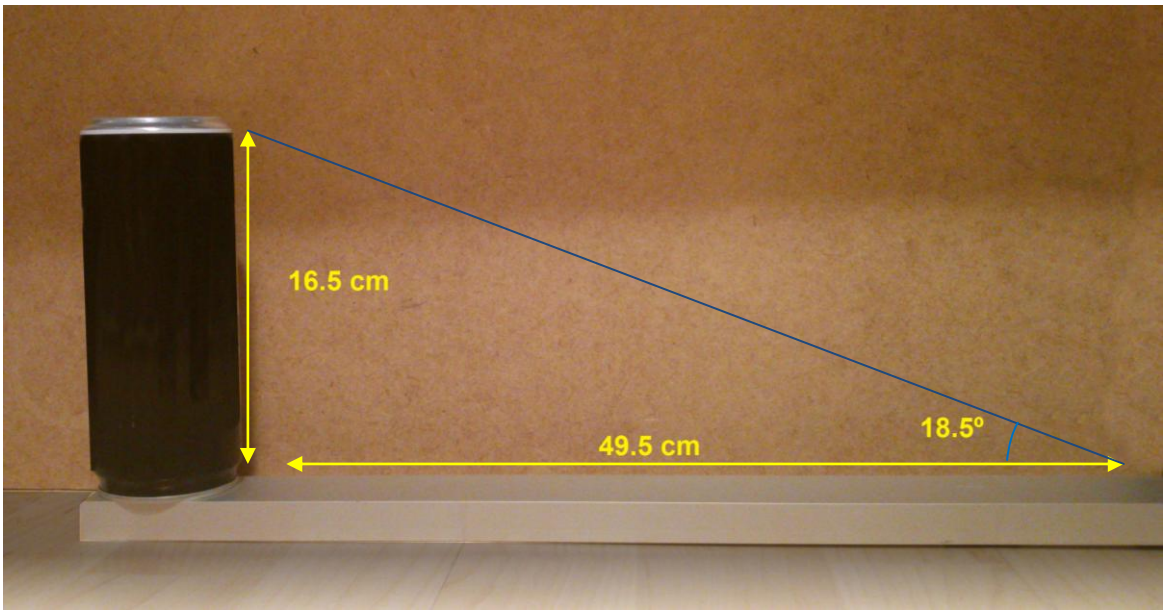


Figura 7. 5: Herramienta diseñada para medir la precisión del sistema estéreo.

Las dimensiones del instrumento no son importantes en sí mismas, basta con que sean conocidas *(Figura 7.6)*.

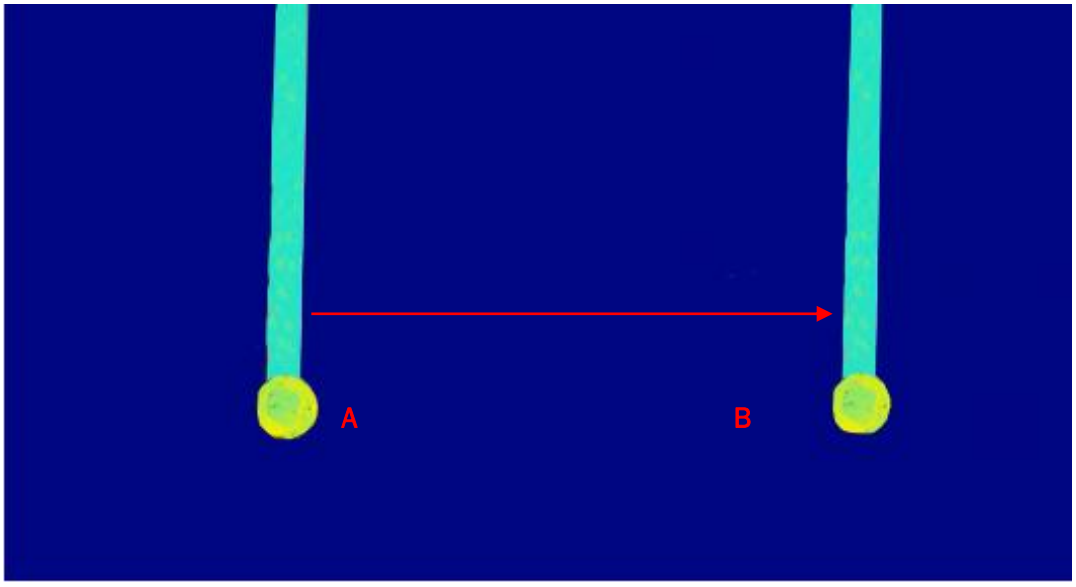


Figura 7. 6: Mapa de profundidades del recorrido realizado con la herramienta de pruebas.

Experimento 1:

Movimiento del instrumento: de manera horizontal, pegado a la mesa y con velocidad constante.

Distancia horizontal entre posición inicial y final: 80 cm

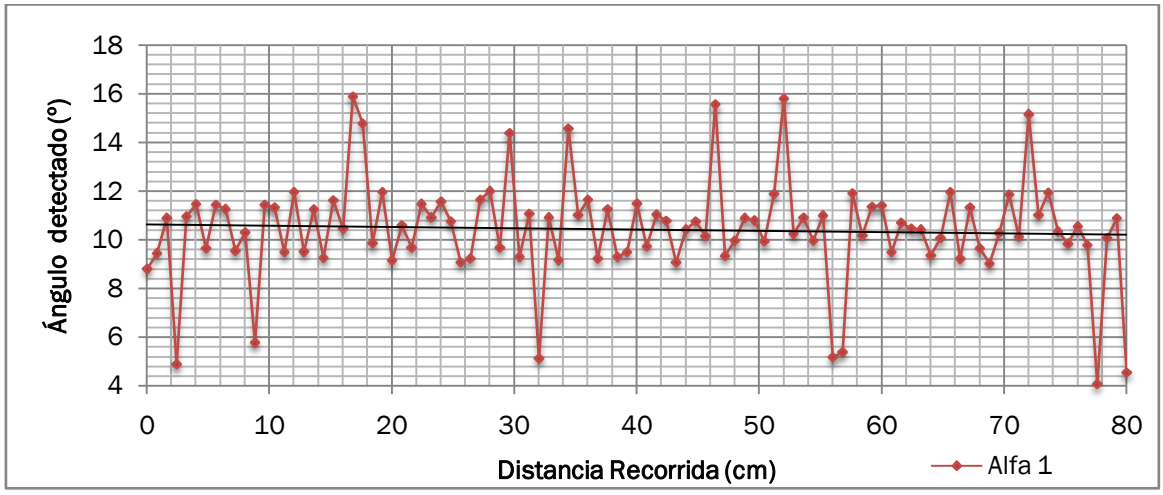


Figura 7. 7: Ángulo Alfa1 de la recta calculada en función de la distancia horizontal recorrida.

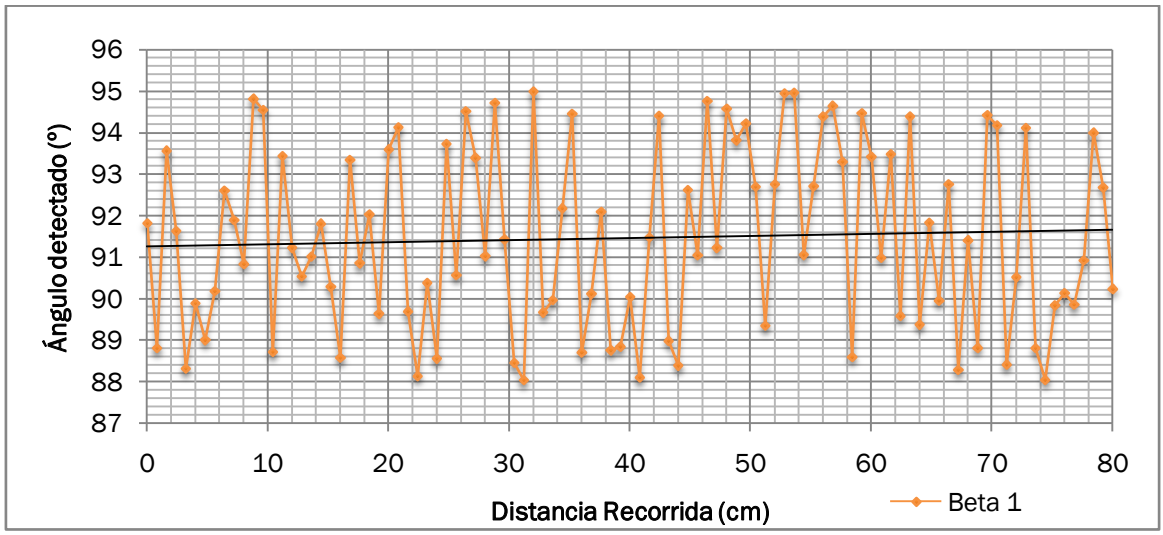


Figura 7. 8: Ángulo Beta1 de la recta calculada en función de la distancia horizontal recorrida.

El ángulo **β**1 da un valor muy aproximado a 90 grados, como era de esperar dado la forma en la que se ha movido el instrumento, sin embargo para el ángulo **α**1 se esperaba un valor aproximado a 18.5 grados dada la geometría de la herramienta. Sin embargo, el valor obtenido oscila en torno a los 11 grados de inclinación *(Figuras 7.8 y 7.9)*.

La explicación para los resultados inesperados en el ángulo $\alpha 1$ es que el instrumento utilizado cuenta con un mango mucho más largo en proporción que la parte elevada, por lo tanto hay muchos más puntos en la zona baja de la nube y eso hace que el ángulo final sea menor de lo que se podría suponer. Esto no es un problema dado que no estamos interesados en obtener un valor absoluto que represente exactamente lo que pasa en la realidad, si no un valor que sirva para identificar la acción.

Experimento 2:

Movimiento del instrumento: estático en el centro de la zona de trabajo y se variará su ángulo elevando únicamente el mango.

Variación de ángulo: 5 grados cuando pase aproximadamente 1 segundo

Duración de la medición: 5 segundos

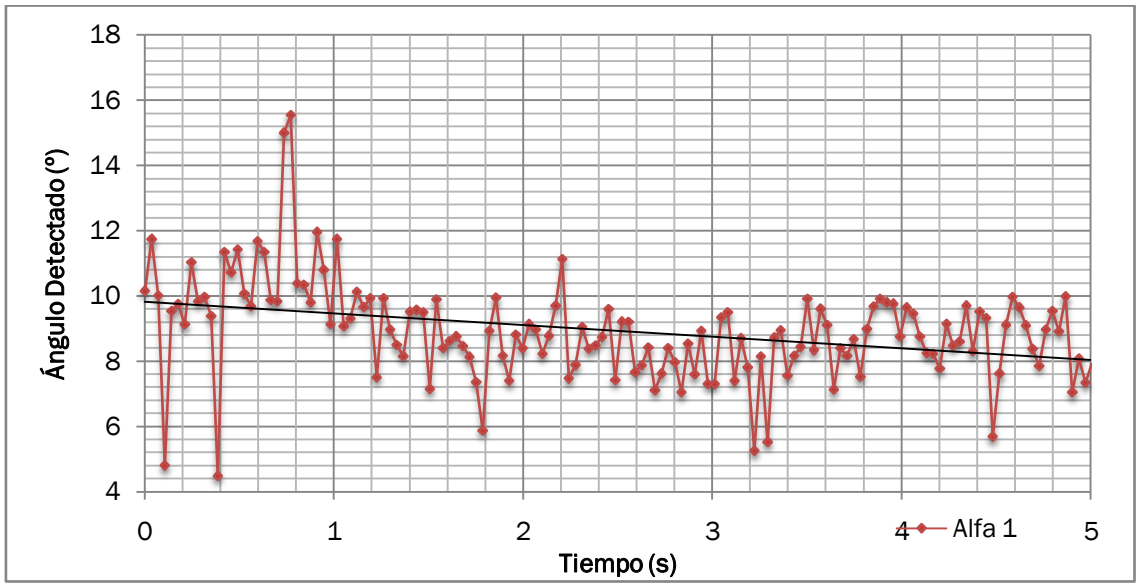


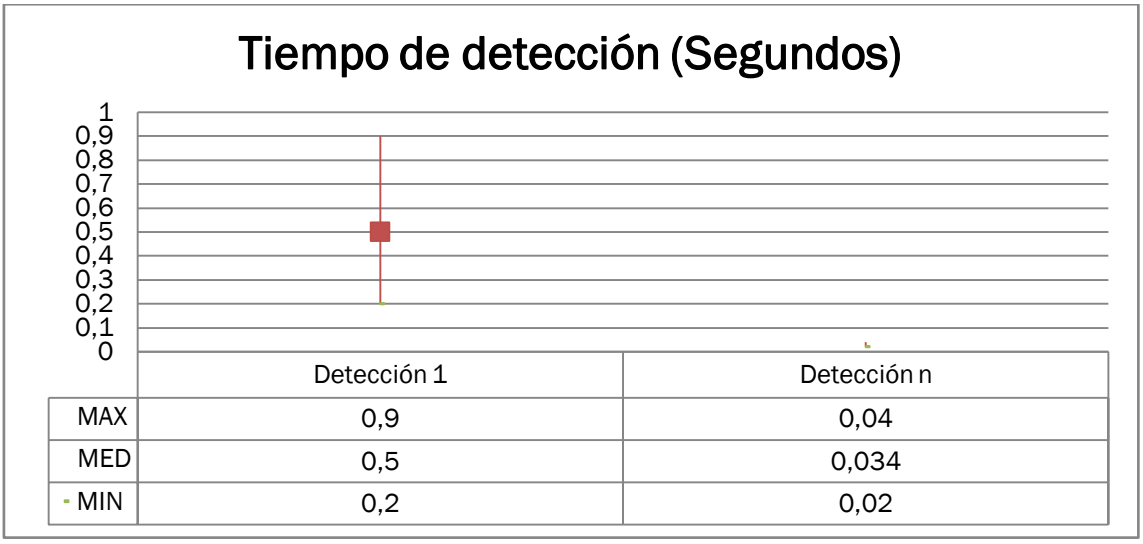
Figura 7. 9: Ángulo Alfa1 de la recta calculada en función del tiempo.

Se puede comprobar que efectivamente, en torno al segundo 1, el sistema detecta un nuevo valor de $\alpha 1$ aproximadamente 2 grados menor que cuando la herramienta estaba completamente horizontal *(Figura 7.9)*.

7.2.2 Plantillas: Tiempos de procesado

A la vista de los resultados obtenidos en el rendimiento del sistema estéreo se decide buscar una alternativa en la estimación de posición mediante marcadores. Como ya se ha visto en el apartado de este trabajo "Estimación de posición a través de plantillas", este método cuenta con la ventaja de ser muy preciso pero la primera detección lleva mucho tiempo.

Para cuantificar esto se decide medir el tiempo que tarda el algoritmo en realizar cada detección en utilizando simplemente una cámara y una plantilla estática situada en el área de trabajo, obteniendo los resultados que se reflejan en la imagen inferior *(Figura 7.10)*.



	Detección 1	Detección n
MAX	0,9	0,04
MED	0,5	0,034
MIN	0,2	0,02

Figura 7. 10: Tiempo de detección del primer fotograma comparado con el del resto de fotogramas.

7.2.3 Conclusión

La visión estéreo funciona bien para variaciones relativamente grandes en la profundidad, no es así con movimientos más leves. Además se puede ver, que aunque la media de los datos obtenidos tiende a un valor concreto, la discrepancia entre dos valores consecutivos de los ángulos puede ser bastante elevada, por lo que no es un buen método para monitorizar pequeños movimientos.

Por otro lado, de los datos obtenidos es fácil concluir que la detección de plantillas no es una opción si la herramienta se mueve muy rápidamente ya que esto equivale a considerar cada imagen como "primera detección", obteniendo tiempos de procesado muy elevados, sin embargo la rapidez y la precisión que se obtienen con este método en movimientos pequeños hace que merezca la pena incluirlo en el algoritmo

La solución a la que se llega después de realizar estos experimentos es utilizar tanto la visión estéreo como la detección por plantillas ya que con el manejo adecuado se complementan para suplir las deficiencias individuales que presentan.

7.3 Tamaño de marcadores

Para analizar el tamaño mínimo en que la detección es efectiva se va a tener que recurrir a la experimentación ya que no se ha encontrado ninguna documentación al respecto.

Uno de los objetivos de este trabajo (fácilmente reproducible) se quiere que cualquiera pueda recrear estos marcadores.

Otro de los objetivos es interferir lo menos posible con el trabajo del operario, por lo que interesa hacer el marcador tan pequeño como se pueda..

Teniendo estas dos cosas en mente se partirá del "Chessboard" oficial de OpenCv y se le reducirá de tamaño utilizando una proporción definida, así cualquiera que quiera utilizar este trabajo puede reproducirlos sin problema.

La lista de proporciones a probar es la siguiente:(1:2),(1:3),(1:4) y (1:5) y para determinar cuál es la óptima simplemente se colocaran todas a rango máximo de las cámaras(directamente encima de la mesa) y se escogerá la más pequeña en la que la detección sea efectiva *(Figura 7.11)*.

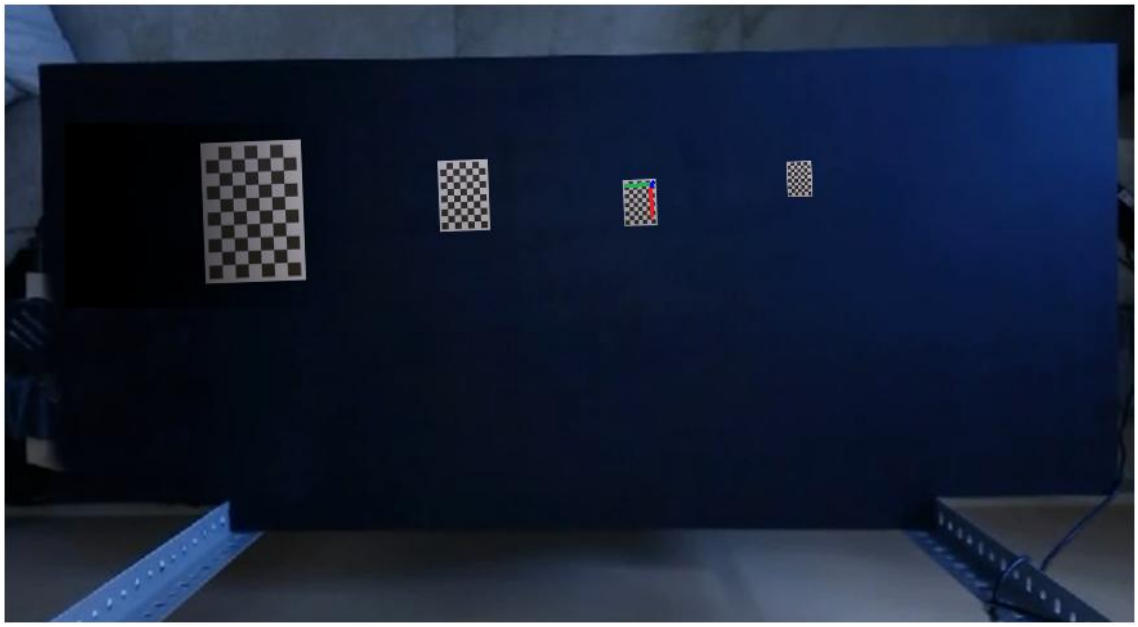


Figura 7. 11: Plantillas de diferentes tamaños utilizadas para determinar el tamaño mínimo posible.

Se determina así que el sistema acepta como máximo trabajar con una escala de (1:4), quedando los marcadores definidos como una sección de 4x3 cuadrados de la plantilla original a esa escala.

8. Uso del sistema

En este capítulo se explica el procedimiento a seguir para hacer funcionar el sistema.

8.1 Definición del área de herramientas

El primer paso consiste en definir el área dentro de la escena donde estarán situadas las herramientas, para eso se ejecutará el programa "CreateROIMask.exe". Al ejecutarse se muestra al usuario la zona de trabajo de tal forma que el usuario defina una región de la misma haciendo click en la imagen *(Figura 8.1)*.

Figura 8. 1: Zona de herramientas señalada en la imagen.

Una vez se esté a gusto con la selección, se utiliza la barra espaciadora para generar la máscara y guardarla en la carpeta del proyecto *(Figura 8.2)*.

Figura 8. 2: Máscara de la zona de herramientas creada a partir de la selección en la Figura 8.1.

8.2 Calibración del estéreo.

Una buena calibración del sistema estéreo es crucial para que los resultados obtenidos sean lo más precisos posibles.

Como ya se ha explicado antes, el objetivo de la calibración es deducir el modelo de las cámaras para poder corregir pequeños errores derivados de posibles distorsiones de la lente, posicionamiento, etc.

Por suerte OpenCV cuenta con herramientas que permiten obtener dicho modelo mediante el análisis de múltiples imágenes en las que aparece un elemento de propiedades conocidas.

En este caso, el objeto de dimensiones conocidas es una plantilla de cuadrados que se puede descargar desde la página oficial de OpenCV, sin embargo se podrían usar también otro tipo de plantillas siempre y cuando el formato esté soportado por OpenCV *(Figura 8.3)*.

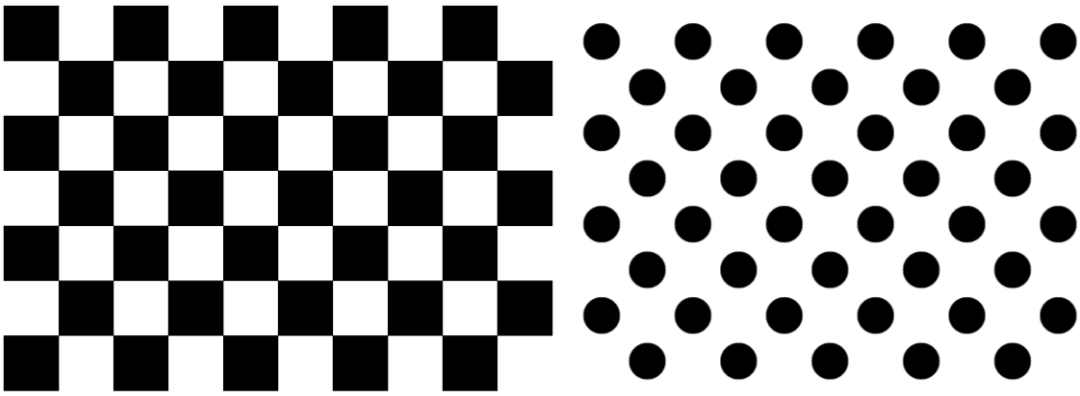


Figura 8. 3: Diferentes plantillas que se pueden utilizar para la calibración del estéreo.

La elección de una u otra plantilla depende tanto de los requerimientos como de los conocimientos del usuario sobre las particularidades de cada una así como de las necesidades y alternativas de las que se disponga.

8.2.1 Toma de imágenes

Una vez decidido qué plantilla se va a utilizar, es necesario adquirir múltiples imágenes de la misma desde ambas cámaras simultáneamente *(Figura 8.4)*.

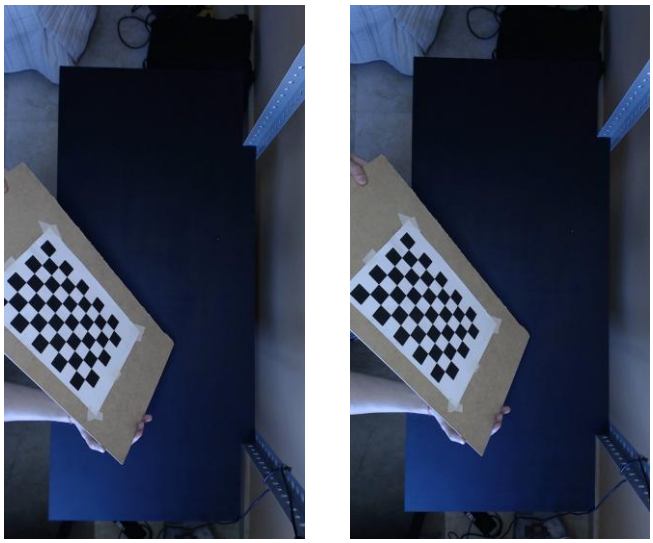


Figura 8. 4: Escena vista desde las dos cámaras.

A la hora de tomar las imágenes es importante tener en cuenta los siguientes factores:

1. A mayor número de imágenes mejor se podrá llevar a cabo la calibración.
2. Cuanto mayor número de posiciones y orientaciones de la plantilla se capturen, mejores resultados se obtendrán.
3. Para una mayor precisión del sistema, los parámetros de las cámaras (distancia focal, tiempo de exposición, balance de blancos, etc.) deberán ser los mismos que se vayan a utilizar durante la ejecución del sistema.
4. Conviene mantener la plantilla lo más quieta posible mientras se capturan las imágenes para limitar en la medida de lo posible distorsiones debidas al movimiento.
5. Cuanta más resolución se utilice mejores resultados se obtendrán, pero el modelo de cámaras que se calcule sólo podrá ser utilizado si el sistema estéreo trabaja a la misma resolución.
6. El tiempo de procesamiento de las imágenes durante la calibración no es relevante ya que sólo se efectúa una vez y uno se puede sentir tentado de utilizar una resolución de 1920x1080 o superior, sin embargo esto significará que a la hora de monitorizar las operaciones el tiempo de computación del computación del mapa de profundidades sea mayor, así que conviene llegar a una solución de compromiso.

Esta captura de imágenes se efectuará mediante el programa "TakeCalibImages.exe" que se puede encontrar en el CD adjunto.

Dicho programa necesita que se configuren los siguientes parámetros en el archivo de configuración "Configuration.yml" *(Figura 8.5)*.

```
#Datos para la  toma de imagenes
CALIBRATION_IMAGES_FOLDER_PATH: "F:/CalibrationImages"
CALIBRATION_IMAGES_NUMBER: 20
CALIBRATION_BOARD_HEIGHT: 6
CALIBRATION_BOARD_WIDTH: 9
```

Figura 8. 5: Parámetros del fichero "Configuration.yml" necesarios para la calibración.

Como se puede ver es necesario aportar tanto el número de imágenes que se tomaran, como el tamaño del tablero utilizado, así como una ruta donde almacenar dichas imágenes .

Nótese que a parte, "TakeCalibImages.exe" también utiliza las variables comunes a todos los ejecutables del sistema como la resolución con la que se trabaja..

8.3.1.1 Procedimiento de toma de imágenes:

Teniendo en cuenta todo lo anterior, Se ejecutará el programa "TakeCalibImages.exe". Se configurarían individualmente los parámetros de las cámaras si fuese necesario (Tiempo de exposición, foco, etc.) y automáticamente el programa comienza la captura de imágenes, almacenándolas en la ubicación especificada para su posterior análisis *(Figura 8.6)*.

Figura 8. 6: Contenido de la carpeta donde se guardan las imágenes de calibración.

8.2.2 Obtención del modelo de las cámaras

Para la obtención del modelo de las cámaras obviamente será necesario haber efectuado antes la captura de imágenes.

Esta calibración se lleva a cabo mediante el ejecutable "CalibrateCameraParameters.exe" que se puede encontrar en el CD adjunto.

En este caso serán necesarias las siguientes variables del archivo "Configuration.yml" *(Figura 8.7)*.

```
#Datos para la calibración del estéreo
CALIBRATION_IMAGE_LIST_PATH: "F:/CalibrationImages/list.xml"
INTRINSICS_FILE_PATH: "F:/CalibrationImages/intrinsics.yml"
EXTRINSICS_FILE_PATH: "F:/CalibrationImages/extrinsics.yml"
```

Figura 8. 7: Parámetros del fichero "Configuration.yml" necesarios para la obtención del modelo de las cámaras.

Los nombres son bastante auto-explicativos, es necesario especificar una ruta y un nombre de fichero tanto para el archivo de parámetros extrínsecos como intrínsecos, así como la ruta del archivo que contiene la lista de imágenes a utilizar.

8.3.2.1 Procedimiento de obtención del modelo de las cámaras

Se ejecuta " CalibrateCameraParameters.exe" y automáticamente se irán cargando todas las imágenes, el programa detectará aquellas en las que la plantilla se observa de manera correcta y descartará las parejas en las que esto no ocurra.

Cabe destacar que si no es posible detectar la plantilla en una imagen de la pareja automáticamente ambas no serán válidas.

Una vez acabado el proceso se guardarán los archivos de los parámetros intrínsecos y extrínsecos en la ubicación seleccionada. además se podrán ver las imágenes rectificadas, así como las rectas epipolares (verde). La zona en rojo determina el área donde la rectificación se considera válida *(Figura 8.8)*.

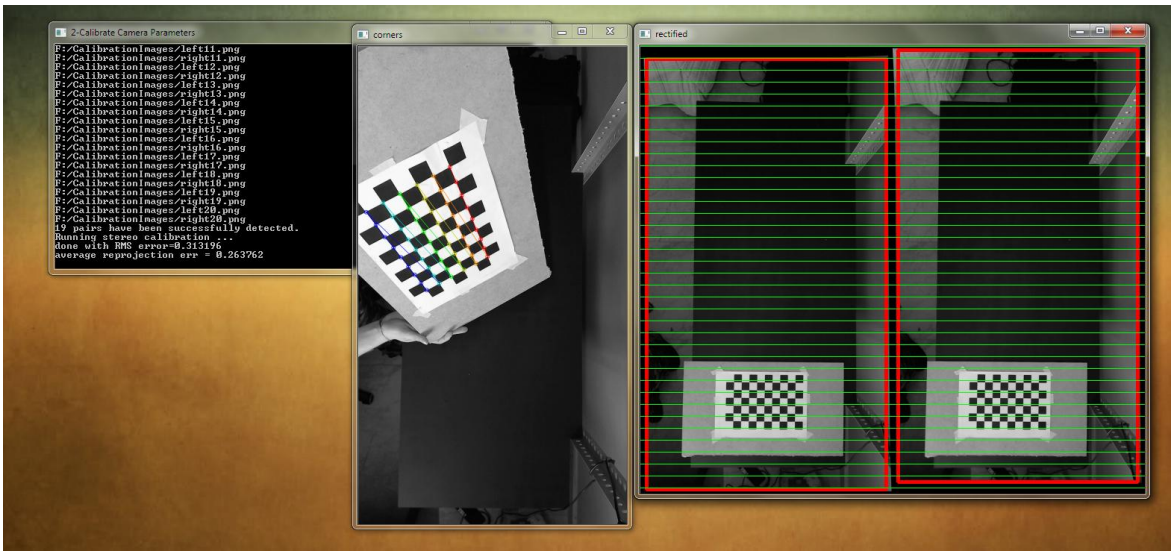


Figura 8. 8: Programa de cálculo de los parámetros del sistema estéreo. Se pueden observar las rectas epipolares halladas (Verde) y la zona de la imagen donde los cálculos son válidos (Rojo)

Si se obtienen errores de reproyección o RMS altos es recomendable tomar un nuevo set de imágenes ya que de la exactitud de este paso depende la exactitud del sistema de visión estéreo.

8.4 Plantillas A-Kaze

Para la detección A-Kaze será necesario aportar imágenes de los objetos que deseamos identificar, para que el algoritmo tenga algo con lo que comparar la imagen recibida.

Con ese propósito será necesario situar dichas plantillas en la carpeta designada para tal propósito. hay que tener en cuenta que el programa cargará las plantillas una única vez al inicializarse para agilizar el proceso, por lo que estas plantillas no deben ser alteradas mientras se está ejecutando el programa *(Figura 8.9)*.

Figura 8. 9: Plantillas utilizadas para la detección de herramientas.

A la hora de crear dichas plantillas, es conveniente mostrar los objetos de la manera más parecida posible a cómo serán vistos en la realidad, es decir, si por ejemplo se trata el caso de un destornillador, conviene que también aparezca la mano del usuario sujetándolo, etc, para maximizar el parecido con la realidad.

También será necesario que dichas plantillas se nombren con el nombre de la herramienta que representan ya que el programa utilizará el nombre de archivo de la plantilla detectada para mostrarlo en pantalla.

8.5 Monitorización de operaciones (programa principal)

Ya sabemos que un aspecto clave para el correcto funcionamiento del sistema consiste en escoger unos parámetros adecuados en la captura de imágenes, por ello, al ejecutar el programa principal "StereoGpu.exe" nos encontramos con que se crea una ventana para cada cámara y se pide que se configuren antes de continuar.

En este paso abriremos el programa "CamControl.exe" y se modificarán los parámetros hasta que se esté a gusto con la configuración obtenida *(Figura 8.10)*.

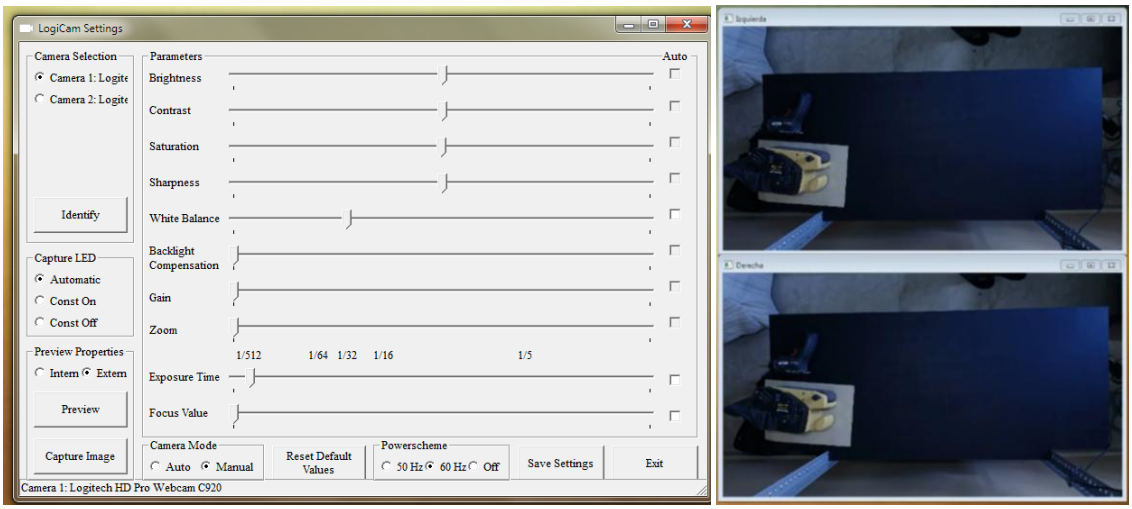


Figura 8. 10: Procedimiento de configuración de los parámetros de adquisición de imagen.

En líneas generales nos interesará que las cámaras no efectúen ningún tipo de autoajuste ya que es necesario tener una imagen lo más similar posible en ambas cámaras.

Para ello se desactivará el ajuste automático tanto de la distancia focal, como del tiempo de exposición.

El resto de parámetros quedan a disposición y juicio del usuario, que decidirá qué es lo que mejor se ajusta a su caso.

Cuando se haya acabado de configurar las cámaras se presiona "S" para continuar.

A partir de este momento el algoritmo toma el mando, en la pantalla se podrá ver el mapa de profundidades de la acción principal. En este momento el programa está

esperando a que el usuario tome una herramienta (se detecte movimiento en la zona de herramientas) para comenzar la monitorización *(Figura 8.11)*.

Figura 8. 11: (Izquierda) Imagen recibida. (Derecha) Mapa de profundidades segmentado a la acción principal.

En cuanto el usuario entra en la zona de las herramientas, el módulo Kaze toma el control e identifica la herramienta seleccionada *(Figura 8.12)*.

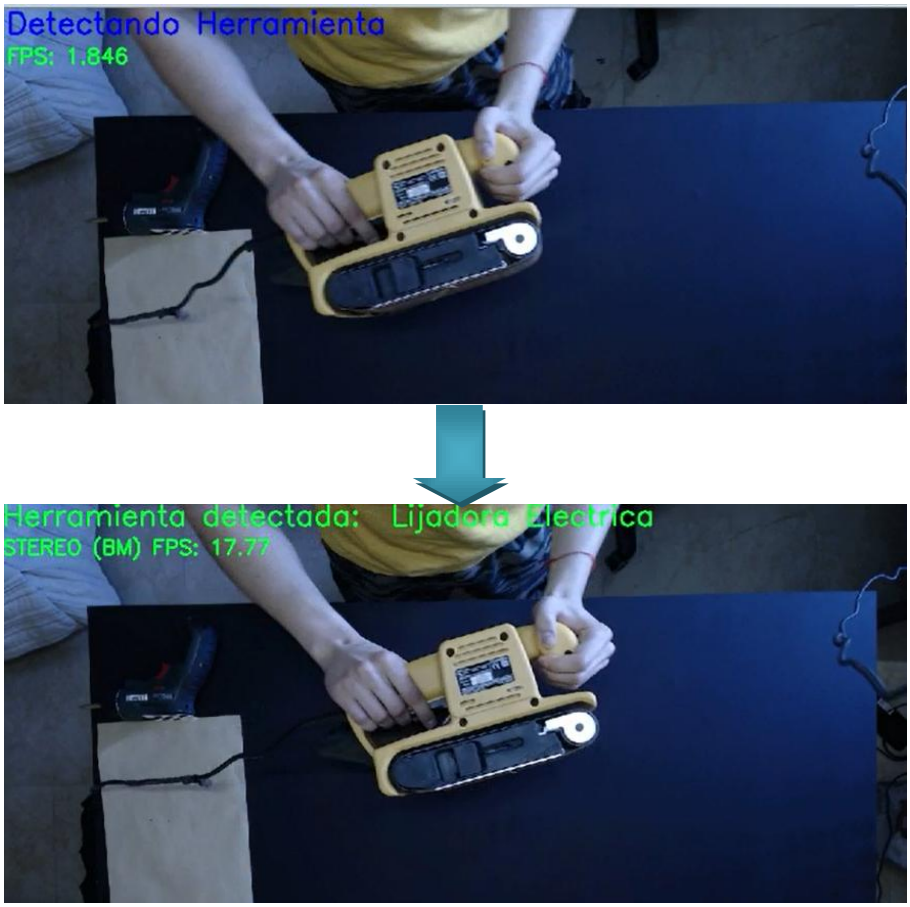


Figura 8. 12: Reconocimiento de la herramienta utilizada cuando el usuario escoge una nueva.

Cuando la detección se ha llevado a cabo de manera correcta se vuelve al módulo de seguimiento, dicho módulo seguirá realizando el seguimiento estéreo hasta que el número de pixeles en la imagen de la acción principal sea lo suficientemente pequeño como para determinar que se está llevando una operación delicada, pasando en ese momento a efectuar una detección mediante plantilla *(Figura 8.13)*.

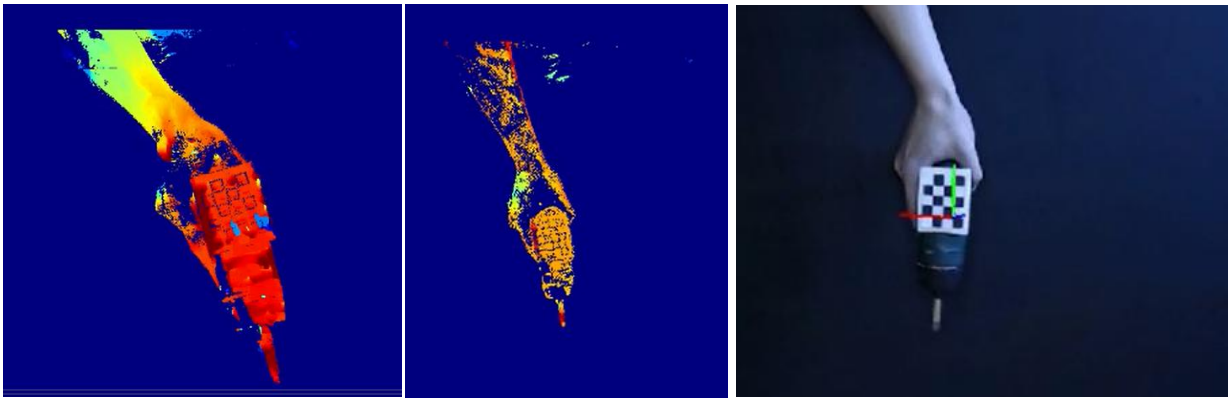


Figura 8. 13: Reducción progresiva del número de pixeles detectados en el sistema estéreo cuando el movimiento es pequeño (primeras dos imágenes) y cambio a detección por plantilla cuando sobrepasa cierto umbral (última imagen.)

9. Estudio Económico

En este capítulo se van a desglosar los detalles sobre los diferentes costes del proyecto completo con el objetivo de determinar tanto viabilidad del mismo como si supone una ventaja económica en comparación con otras posibles soluciones.

A tal efecto será necesario estudiar todos los costes del proyecto, esto incluye tanto costes directos (materiales) como indirectos (mano de obra, instalación…etc) y por otros costes como pueden ser gastos de desplazamiento electricidad.

9.1 Recursos empleados

En todo proyecto tecnológico hay una serie de recursos tanto físicos como software que son utilizados en las diferentes etapas del mismo. Aquí se incluyen los ordenadores, sus sistemas operativos o incluso el material de oficina, por ejemplo.

A la hora de escoger las herramientas a utilizar es muy útil tener en cuenta la existencia de herramientas de código abierto, este tipo de sistemas reducen enormemente los costes de un proyecto al proveer los mismos servicios que sus equivalentes comerciales, pero con coste cero. Un ejemplo de este tipo de recursos es la librería utilizada (OpenCV), que es de código abierto y permite su uso o incluso comercialización de manera libre.

A continuación se puede encontrar una liste de todos los recursos utilizados para el desarrollo de este proyecto, independientemente de si suponen un coste..

- Sistema operativo: Windows 7 Professional
- Entorno de programación: Microsoft Visual Studio 2013
- Librería de visión por computador: OpenCV
- Otros programas: paquete ofimático Microsoft Office 2007.
- Ordenador: portátil Mountain StudioMx.154G
- Cámaras: Logitech C920
- Material ofimático: libros de consulta y otros consumibles (folios A4, bolígrafos, lapiceros…).

9.2 Costes directos

Entre los costes directos se incluyen aquellos imputables directamente al desarrollo del presente proyecto, como puede ser la mano de obra utilizada, los materiales utilizados únicamente en el sistema desarrollado o la amortización de equipos y programas en teniendo en cuenta el tiempo que se han estado utilizando para el desarrollo del sistema en comparación con su vida útil total.

9.2.1 Costes de personal

El proyecto ha sido llevado a cabo en su totalidad por un ingeniero, que ha sido encargado de las investigaciones previas, el diseño y la implementación de todo el sistema de visión artificial.

Para contabilizar estos costes, en primer lugar hay que tener en cuenta el salario, tanto bruto anual como las cotizaciones a la Seguridad Social (35% del sueldo bruto) y los días que este salario representa. Estos datos se pueden encontrar reflejados en la tabla 9.1.

COSTE ANUAL	
Sueldo bruto más incentivos	35.000,00 €
Seguridad Social (35% sueldo bruto)	12.250,00 €
Coste total	47.250,00 €

Tabla 9. 1: Salarios

En segundo lugar, los días de trabajo, se representan en la *tabla 9.2*:

DÍAS EFECTIVOS POR AÑO	
Año medio	365,25 días
Sábados y Domingos	-104,36 días
Días de vacaciones efectivos	-20,00 días
Días festivos reconocidos	-15,00 días
Días perdidos estimados	-5,00 días
Total días efectivos estimados	220,89 días

Tabla 9. 2: Días de trabajo

Con el número de días, y teniendo en cuenta una jornada laboral de 8 horas, el cálculo del **total de horas efectivas de trabajo** en un año es:

$$220{,}89\frac{días}{año}\ x\ 8\frac{horas}{día} = \mathbf{1.767{,}12}\frac{\boldsymbol{horas}}{\boldsymbol{año}}$$

Con el sueldo anual anteriormente calculado y estas horas efectivas de trabajo, se puede obtener el coste por hora de un ingeniero:

$$\frac{47.250\,\frac{€}{año}}{1.767{,}12\,\frac{horas}{año}} = \mathbf{26{,}73}\ €/\boldsymbol{hora}$$

En la siguiente tabla (*tabla 9.3*) se muestra la distribución temporal aproximada del trabajo de ambos ingenieros en el presente proyecto, en un desglose de las horas de cada una de las partes de que consta el mismo:

DISTRIBUCIÓN TEMPORAL DE TRABAJO	
Formación y documentación	50 horas
Estudio del problema	50 horas
Desarrollo de la aplicación	100 horas
Puesta a punto del sistema	70 horas
Elaboración de la documentación	150 horas
Total de horas empleadas	420 horas

Tabla 9. 3:. Distribución temporal de trabajo.

Por último, para calcular el coste de mano de obra imputable al proyecto, es decir, el **coste personal directo**, se multiplican las horas empleadas por los ingenieros por el coste por hora calculado anteriormente, obteniéndose:

$$420\ horas\ x\ 26{,}73\,\frac{€}{hora} = \mathbf{11.226{,}60}\ €$$

COSTE PERSONAL DIRECTO	11.226,60 €

9.2.2 Costes de amortización de equipos y programas

Aquí se incluyen los costes del uso de los diferentes equipos y programas utilizados que tienen una vida útil mayor a la del presente trabajo, obviamente excluyendo el uso de programas de código abierto (OpenCV) y los de coste cero (Microsoft Visual Studio).

Se empieza calculando el tiempo de amortización de cada elemento, que es la vida útil estimada del correspondiente material. A partir de ese dato se puede calcular un factor que posteriormente multiplique al coste real del material en concreto para hallar su coste dentro del proyecto.

Consideramos que tanto Ordenador como Ratón tienen una vida útil de 5 años. de esta forma su factor de amortización es 0.20.

Microsoft Windows 7 y Microsoft Office 2007: vida útil, 10 años, siendo el factor de amortización 0.1.

En la tabla 9.4 se pueden ver los costes de amortización de equipos:

MATERIAL	IMPORTE	FACTOR DE AMORTIZACION	AMORTIZACION ANUAL
Portátil Mountain StudioMX 154g	1363€	0.20	272.6€
Ratón Razer Dathadder	70€	0.20	14€
Microsoft Windows 7	137,00 €	0.10	13.70€
Microsoft Office 2007	229,00 €	0.10	22.90€
TOTAL			323.20€

Tabla 9. 4: Amortización de material.

El coste final por hora de utilización del material es calculado mediante la división de la amortización anual entre el número de horas de uso en dichos equipos, es decir, el número de horas efectivas de trabajo en un año:

$$\frac{323{,}2\,\frac{€}{año}}{1.767{,}12\,\frac{horas}{año}} \simeq \mathbf{0{,}183}\ €/\boldsymbol{hora}$$

Por último, para averiguar el coste real de amortización del material se multiplica el coste por hora por el número total de horas que se han utilizado los equipos y programas mencionados, en todas las etapas de desarrollo del proyecto:

$$420\ horas\ x\ 0{,}183\,\frac{€}{hora} = \mathbf{76{,}81}\ €$$

COSTE DE AMORTIZACIÓN DE MATERIAL DE EQ. y PROG.	76,81 €

9.2.3 Costes derivados de otros materiales

En estos costes se incluyen los costes de todo tipo de consumibles utilizados en la elaboración del proyecto y sus documentos a lo largo de toda la fase del desarrollo, como por ejemplo papel, tinta, coste de fotocopias externas, almacenamiento de documentos y programas, etc.

También se ha incluido aquí el coste de las cámaras utilizadas ya que son parte vital y necesaria del sistema.

En la *tabla 9.5* se realiza una estimación de algunos de estos costes.

MATERIAL	IMPORTE
Logitech C920 (2)	127.90€
Fotocopias	20,00 €
Almacenamiento	10,00 €
Otro material de oficina	20,00 €
Total material consumible	177,90 €

Tabla 9. 5: Coste del material consumible.

COSTE DE MATERIALES CONSUMIBLES	177,90 €

9.2.4 Costes directos totales

Para hallar los costes directos totales, se suman todos los costes directos anteriores, es decir, personal, amortización de equipos y otros materiales.

11.226,60 € + 76,81€ + 177,90 € = ***11.481.31 €***

COSTES DIRECTOS	**11.481,31 €**

9.3 Costes indirectos

Los costes indirectos son aquellos gastos producidos por la actividad asociada al proyecto, pero que no se pueden imputar directamente al mismo exclusivamente porque pueden estar involucrados en otros procesos o proyectos.

En la *tabla 9.6* se desarrollan estos gastos:

COSTES INDIRECTOS PARCIALES	
Dirección y servicios administrativos	150,00 €
Consumo de electricidad	100,00 €
Consumo de desplazamiento	50,00 €
Total gastos indirectos	300,00 €

Tabla 9. 6: Costes indirectos.

Por tanto, los costes indirectos totales ascienden a:

COSTES INDIRECTOS	**300,00 €**

9.4 Costes totales

Los costes totales son el resultado de sumar los gastos directos e indirectos, siendo el montante total para este proyecto:

COSTES TOTALES	
Costes directos	11.481,30 €
Costes indirectos	300,00 €
Coste total del proyecto	11.781,30 €

Tabla 9. 7: Costes totales

En definitiva, el coste del proyecto es:

COSTES TOTALES DEL PROYECTO	**11.7981,30 €**

10. Conclusiones y trabajos futuros

La idea inicial tras este proyecto, residía en llevar a cabo un sistema que fuese capaz de monitorizar ciertas tareas llevadas a cabo por un operario dentro de un espacio de trabajo limitado.

Cuando se conocen exactamente la naturaleza y las características de las tareas a monitorizar, es posible diseñar un sistema que se adapte muy bien a las mismas. Sin embargo este no ha sido el escenario en el que se ha tenido que trabajar, así que se ha optado por elevar el nivel de abstracción y diseñar un sistema que pueda ser utilizado para seguir operaciones independientemente de la naturaleza de las mismas.

Ante un problema tan amplio no es sencillo encontrar una solución perfecta cuando el tiempo del que se dispone es limitado, por ello y como punto fuerte de este sistema se ha dedicado un esfuerzo especial en crear una base sólida que pueda ser utilizada como punto de partida en futuros trabajos minimizando así el grado de redundancia entre trabajos y en última instancia conseguir alcanzar objetivos más complejos.

Cada decisión tomada a lo largo de este trabajo ha tenido siempre el objetivo de facilitar la compatibilidad y expansibilidad del mismo. La librería elegida (OpenCV) cuenta con una licencia BSD que no sólo permite ser utilizada de manera libre, si no que además se pueden *comercializar* las aplicaciones que la utilicen. Esta característica hace que sea una opción excepcional para reducir los costes del sistema ya que de utilizar librerías comerciales, es habitual tener que su uso y comercialización supongan un coste mensual. Además, OpenCV cuenta con gran comunidad detrás gracias a la cual el aprendizaje resulta sencillo.

De la misma forma, otra medida que reduce los costes y aumenta la facilidad de uso es la elección de cámaras de consumo (Logitech C920) en vez de sus equivalentes profesionales mucho más caras.

Por otro lado el uso de CUDA, aunque sacrifica un poco de compatibilidad limitando el uso del sistema a máquinas con tarjetas gráficas Nvidia, lo dota de la capacidad para adaptarse a tareas que requieren un elevado poder de computación a la vez que optimiza la implementación de algoritmos que por su naturaleza no son tan efectivos cuando se llevan a cabo en un procesador, como es el caso de la visión estéreo.

Uno de los logros de este sistema consiste en la integración de diferentes técnicas utilizadas habitualmente en la visión por computador de manera que se complementen logrando mejores resultados. De esta manera, mediante el uso conjunto la visión estéreo y la estimación de posición a través de marcadores, se ha conseguido trabajar con una resolución más elevada que lo que hasta ahora se había conseguido en anteriores trabajos (720p) analizando una media de aproximadamente 25 FPS.

Eso ha sido posible gracias al estudio en profundidad de los algoritmos disponibles y gran cantidad de pruebas llevadas a cabo para determinar los puntos fuertes y los cuellos de botella de cada uno hasta lograr una combinación que cumpliese las expectativas.

Un ejemplo de esto se puede ver en el algoritmo escogido para la detección de herramientas (KAZE). Ser capaz de detectar objetos independientemente de su orientación utilizando puntos clave tiene un coste computacional elevado y reduce mucho la cantidad de fotogramas que se analizan cada segundo, sin embargo utilizando técnicas de segmentación que permiten ejecutarlo sólo cuando es necesario (el usuario coge una herramienta) ha sido posible contar con detección de objetos efectiva sin comprometer la usabilidad del sistema completo.

Por otro lado, enfocar los esfuerzos en la línea de la compatibilidad ha provocado que se dejasen un poco de lado aspectos tales como crear una interfaz gráfica más fácil de utilizar que la línea de comandos, sin embargo se considera que ya existe suficiente información en ese aspecto y que merecía le pena dedicar más trabajo a otras áreas.

Otra de los puntos en el que se ha que se han cedido al escoger esta línea de trabajo es que el sistema requiere una mayor interacción por parte del usuario con en forma de configuración y adaptación a su uso, sin embargo en general las ventajas de el enfoque seguido superan ampliamente estos inconvenientes y los resultados obtenidos en las pruebas realizadas confirman lo dicho anteriormente.

10.1 Trabajos futuros

Es interesante también explorar posibles usos que se le podrían dar al sistema más allá de aquel para el que fue originariamente concebido así como mejoras que lo dotasen de nuevas características y funcionalidades.

A continuación se pasarán a exponer varios puntos que se podrían considerar como posibles líneas de trabajo para futuros proyectos basados en este trabajo.

Aceptación del sistema: es posible y comprensible que muchos operarios vean el sistema como un instrumento de vigilancia y control por parte de la empresa, causando estrés y rechazo a ser utilizado, por ello es muy importante establecer unos límites sobre lo que el sistema es capaz de hacer o "ver" y garantizar que los operarios conocen plenamente qué datos se están obteniendo de su trabajo para que de esta forma se entienda que el sistema es en realidad un herramienta de apoyo más que de vigilancia.

Ergonomía y seguridad: a parte de efectuar un control de las operaciones per se, se podría aprovechar la información que obtiene el sistema para de alguna manera determinar si el usuario usa las herramientas y los materiales de manera correcta y segura.De esta manera se podrían minimizar los posibles errores y accidentes debidos a un mal uso del equipamiento lo que repercutiría directa y positivamente en la seguridad del usuario.

Formación de nuevos empleados: este sistema puede ser utilizado también en la formación de nuevos operarios, evitando que se adquieran malos hábitos desde un primer momento a la vez que se facilita el aprendizaje.

Guiado de robots: la programación de un robot es una tarea tediosa y que requiere tiempo. Adaptando este sistema se podría utilizar el conocimiento y habilidad de un operario experto para obtener todos los datos relevantes de los movimientos y tareas que lleva a cabo y utilizaros para que un robot o grupo de robots los imiten.

Estimación de productividad: utilizando los datos obtenidos del sistema, sería posible legado el caso obtener unos tiempos teóricos óptimos para cada operación y comprobar que efectivamente el operario está trabajando de la manera que se espera o si podría mejorar su manera de proceder de alguna manera.

BIBLOGRAFÍA

[1] OpenCV. Visitado en Marzo de 2015. http://opencv.org/

[2] Yoke Peng Leong (2014) - Stereo Vision Tracking System Using Webcams. http://www.cds.caltech.edu/~yleong/docs/Research_Reports/CNS186_Final_Report_Stereo.pdf
Visitado en Marzo de 2015

[3] The Random Lab: Logitech c920 and c910 fields of view for RGBD toolkit. http://therandomlab.blogspot.com.es/2013/03/logitech-c920-and-c910-fields-of-view.html
Visitado en Marzo de 2015

[4] Kimchi and Chip's blog: evaluation of logitech c910 for computer vision. http://www.kimchiandchips.com/blog/?p=564
Visitado en Marzo de 2015

[5] Stack Overflow: camera suggestions for machine vision with OpenCV. http://stackoverflow.com/questions/20968399/camera-suggestion-for-machine-vision-with-opencv
Visitado en Marzo de 2015

[6] Guo-Quing Wei and Song De Ma (1994) - Implicit and Explicit Camera Calibration: Theory and Experiments. http://www-dsp.elet.polimi.it/VA-TLC/Articoli/Wei-Ma_94_Calibration.pdf
Visitado en Marzo de 2015

[7]Pablo Fernández Alcantarilla, Adrien Bartoli, Andrew J. Davison2 - KAZE Features. https://www.doc.ic.ac.uk/~ajd/Publications/alcantarilla_etal_eccv2012.pdf
Visitado en Agosto de 2015

[8] Peter Andreas Entschev - Image learning and computer vision in CUDA.
http://on-demand.gputechconf.com/gtc/2015/presentation/S5796-Peter-Andreas-Entschev.pdf
Visitado en Agosto de 2015.

[9] P M Panchal , S R Panchal , S K Shah - Comparison between SIFT and SURF.
http://www.ijircce.com/upload/2013/april/21_V1204057_A%20Comparison_H.pdf
Visitado en Septiembre de 2015.

[10] Luo Juan - A Comparison of SIFT, PCA-SIFT and SURF.
http://www.cscjournals.org/manuscript/Journals/IJIP/volume3/Issue4/IJIP-51.pdf
Visitado en Septiembre de 2015

[11] Can Erodgan, Manohar Paluri, Frank Dellaert - Planar Segmentation of RGB Images using Fast LinearFitting and Markov Chain Monte Carlo.
http://www.cc.gatech.edu/~dellaert/pub/Erdogan12crv.pdf
Visitado en Septiembre de 2015

[12] Ninad Thakoor, Sungyong Jung, Jean Gao - Real-time Planar Surface Segmentation in Disparity Space.
http://www.ece.ucr.edu/~nthakoor/Papers/ECVW.pdf
Visitado en Septiembre de 2015

[13] A Short Course Book Stereo Photography 3D in the Digital Era - Understanding the Baseline. http://www.shortcourses.com/stereo/stereo3-14.html
Visitado en Septiembre de 2015

[14] A Eduardo Ojosnegros Ramos - Desarrollo de un Sistema para la Monitorización de Operaciones Manuales Mediante Sensor Kinect

[15] Speed Up Robust features. http://mokga.tistory.com/entry/SURFSpeed-Up-Robust-Features
Visitado en Septiembre de 2015

[16] Surveon 3 MP 30 FPS H.264 HDR Camera Series.
http://www.slideshare.net/Surveon/surveon-3-mipcamv06eng
Visitado en Septiembre de 2015

[17] Camera calibration With OpenCV.
http://docs.opencv.org/doc/tutorials/calib3d/camera_calibration/camera_calibration.html
Visitado en Septiembre de 2015

[18] Nobuyuki Otsu: A threshold selection method from grey level histograms. In: IEEE Transactions on Systems, Man, and Cybernetics. New York 9.1979, S.62–66. "ISSN 1083-4419.

[19] CUDA: Parallel Programming and Computing Platform.
http://www.nvidia.com/object/cuda_home_new.html
Visitado en Septiembre de 2015

[20] Harris Corner Detection.
http://www.cse.psu.edu/~rtc12/CSE486/lecture06.pdf
Visitado en Septiembre de 2015

ANEXO: CONTENIDO DEL CD:

En el CD está incluida la vesión en PDF de este trabajo, así como la solución para Microsoft Visual Studio 2013 con el código fuente.

También se incluyen los archivos necesarios para la ejecución del programa "out of the box".

CONTENIDOS DEL CD