UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

TRABAJO FIN DE MASTER EN INGENIERÍA INDUSTRIAL

DISEÑO, DESARROLLO Y EVALUACIÓN DE UN SISTEMA DE SEGUIMIENTO VISUAL DE ALTA VELOCIDAD

AUTOR: GUIÑÓN CATALÁN, JUAN JOSÉ

TUTOR: SÁNCHEZ SALMERÓN, ANTONIO JOSÉ

Curso Académico: 2016-17

RESUMEN

El presente trabajo se centra en el desarrollo y evaluación de un sistema de seguimiento de objetos en el espacio mediante visión artificial estereoscópica adaptado para alta velocidad en aplicaciones del campo de la robótica y automatización. El seguimiento de objetos mediante imagen estereoscópica es un campo ampliamente estudiado y del que se dispone gran cantidad de material, como demuestra la existencia de librerías especializadas de dominio público en internet. No obstante, dadas las exigencias a nivel computacional, la detección de objetos está sometida a una importante limitación a nivel de velocidad. Este trabajo parte de un algoritmo básico de detección de una pelota roja y su triangulación en el espacio mediante geometría epipolar y analiza diversos métodos de optimización para acelerar el procesado de las imágenes. Los métodos estudiados son el uso de modelos del movimiento del objeto a seguir para acotar la zona de detección y modelos de predicción para compensar posibles fallos en la detección bien asumiendo el objeto como estático cuando se pierde o bien con un filtro de Kalman para la predicción de la trayectoria. Se comparan como afectan dichos algoritmos tanto a nivel de imágenes procesadas por segundo como por sus efectos en la precisión de las medidas. De este análisis se determina como factor principal en la velocidad de procesado el área de búsqueda en la detección geométrica. Este estudio pretende establecer una base en cuanto a eficacia y eficiencia para métodos de optimización de visión artificial en posibles futuras aplicaciones.

Palabras Clave: “Visión artificial”, “detección de objetos”, “seguimiento de objetos“, “alta velocidad”.

ÍNDICE

MEMORIA

1. Objetivo

En el presente trabajo de fin de máster se diseñará un prototipo de visión automático para el seguimiento de objetos en el espacio mediante visión estereoscópica utilizando dos cámaras de alta velocidad.

El sistema estará compuesto por un PC, una tarjeta de captura de imágenes y dos cámaras. Se utilizará el lenguaje de programación C++ y las librerías de código abierto OpenCV.

Se evaluará el prototipo en función tanto de la precisión de las medidas como de la velocidad de las mismas.

2. Introducción

2.1. Contexto

Este sistema estará enfocado hacia el ámbito de la robótica, en el que el uso de sistemas de visión artificial para el control de brazos robot en aplicaciones de manipulación de objetos[1] pero también en multitud de aplicaciones[2] en las que se tiene menor control sobre el objeto a detectar o se necesite conocer su forma para poder manipularlo en consecuencia.

2.2. Antecedentes

El campo del seguimiento de objetos mediante visión artificial está ampliamente estudiado, y por consiguiente existen numerosas herramientas disponibles para ello. Parte de este trabajo se fundamenta en estudios previos[3] en el campo de la mejora del seguimiento de objetos mediante técnicas de predicción.

Desde el punto de vista de la implementación, también existe una amplia variedad de materiales. Las librerías de código abierto OpenCV, desarrolladas de manera desinteresada por numerosos autores, (en la versión 3.2.0 durante el desarrollo de este trabajo) disponen de códigos altamente optimizados específicamente tanto para la detección de objetos como para la triangulación mediante imagen estereoscópica.

2.3. Fundamentos teóricos

Para el desarrollo del proyecto se ha escogido utilizar las librerías de visión artificial de OpenCV por su asequibilidad y disponibilidad de documentación en la red. OpenCV es un conjunto de librerías originalmente desarrolladas por la empresa Intel en 1999. Está publicada bajo licencia BSD permitiendo su uso tanto en aplicaciones comerciales como de investigación.
OpenCV está programado en C/C++, con un alto nivel de optimización y dispone de gran variedad de funciones de procesado de imagen de forma eficiente a nivel computacional descritas ampliamente en la página oficial de desarrollo[4] [5]. Entre ellas las empleadas para el desarrollo de este proyecto que se describen a continuación:

2.3.1. Detección de bordes de Canny

El detector de bordes de Canny fue desarrollado por John F. Canny en 1986 y se trata de un método altamente eficaz en la detección de contornos debido a su bajo ratio de errores, alta precisión, y respuesta mínima, dando un único valor por borde.

El método que emplea es el siguiente:

En primer lugar se filtra el ruido mediante un filtro gaussiano. A continuación se busca la intensidad de gradiente de la imagen utilizando un procedimiento análogo al filtro Sobel. Se aplica una máscara de convolución en los ejes *x* e *y*:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix};\ G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

Ec 1. Máscaras de convolución

Y se busca el módulo y dirección del gradiente, redondeando el ángulo a múltiplos de 45°

$$G = \sqrt{G_x^2 + G_y^2};\ \theta = \arctan\left(\frac{G_y}{G_x}\right)$$

Ec 2. Cálculo del gradiente

A continuación se suprimen todos los valores no máximos. De esta forma se eliminan todos los píxeles que no son parte de un borde.

Por último se aplica un filtrado con histéresis. Se toman dos valores umbral, los gradientes de valor mayor al umbral superior se consideran bordes, los valores inferiores al umbral menor no se consideran bordes, y los valores intermedios se consideran bordes si están conectados a píxeles que se consideren borde.

Esta función no se aplica independientemente, sino que forma parte de la función de detección de contornos mediante transformada de Hough, explicada a continuación.

2.3.2. Transformada de Hough

La transformada de Hough es un método muy extendido de detección de líneas[6], basado en un sistema de votación por puntos, que resulta especialmente práctico para el procesado de imágenes digitales.

El sistema emplea un barrido de posibles líneas determinadas en coordenadas polares, contando cuantos puntos de los que se quieren analizar coinciden con ellas. De este modo se obtiene una matriz de "votación" en la que se tiene un recuento de cuantos puntos pasan por cada línea. Las dimensiones de esta matriz dependen de la resolución que se desee a la hora de reconstruir las líneas, teniendo en cuenta que mayor resolución implica un tiempo de cálculo considerablemente mayor, y menor resolución conlleva menor ajuste de los resultados obtenidos a las líneas reales.

De estas matrices se obtienen los máximos, que representan las definiciones de las líneas más probables en la imagen.

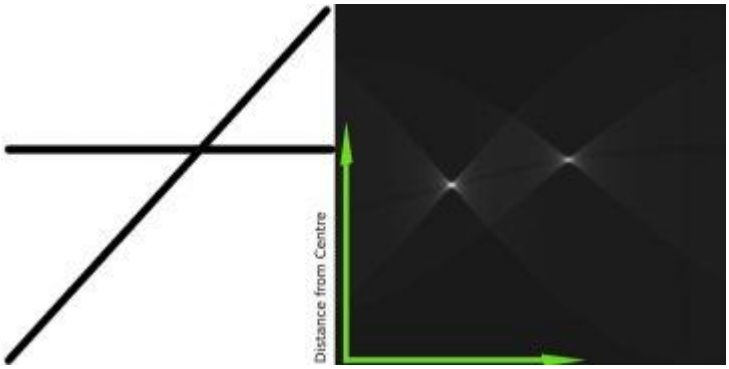


Fig. 1. Líneas y representación visual de la matriz obtenida por transformada de Hough. Los puntos más brillantes representan máximos. Los ejes representan las coordenadas polares de rectas, ángulo en el eje horizontal y distancia en el eje vertical

http://docs.opencv.org/trunk/houghlines2.jpg

Para la detección de círculos, OpenCV no utiliza la transformada de Hough mediante puntos sino mediante gradientes, por este motivo, el filtro Canny no se utiliza de forma independiente sino dentro de la función.

Se emplean gradientes para la detección de formas cerradas debido a que proporcionan una cantidad mucho menor de falsos positivos. Esto se debe a la información extra que aportan: en una figura determinada por una línea cerrada, los gradientes de todo el contorno tienen el mismo signo, es decir, todos apuntan o bien hacia el interior de la figura o hacia el exterior.

La función específica de detección de círculos devuelve las coordenadas del centro de los círculos más probables, así como su radio. La función permite acotar la búsqueda estableciendo un radio mínimo y máximo de los círculos que se pretende encontrar y la distancia mínima entre centros.

2.3.3. Filtro de Kalman

El filtro de Kalman es un algoritmo de identificación de la posición de un sistema dinámico lineal, teniendo en cuenta un posible ruido blanco en las medidas. Fue desarrollado en 1960 por Rudolf Kalman y cuenta con numerosas aplicaciones, especialmente en el campo de la navegación, pero también en el campo de la robótica[7].

Se trata de un filtro recursivo, que puede utilizarse en tiempo real, utilizando las medidas aportadas, el ultimo valor obtenido y la matriz de incertidumbre que modela las varianzas de los errores que causa el ruido que afecta al sistema.

Para el estudio se emplea un filtro de Kalman con tiempo discreto, debido a que los valores obtenidos se limitan a instantes en los que se realiza la medición.

$$x_k = A_{k-1}x_{k-1} + B_{k-1}u_{k-1} + w_{k-1}$$
$$z_k = H_k x_k + v_{k-1}$$

Ec 3.1 y 3.2. Filtro de Kalman

Donde w_k es un ruido blanco de media cero y varianza Q_k en el instante *k*, v_k es un ruido blanco distinto y con varianza R_k en el instante *k*. El filtro estima el estado x_k a partir de las medidas anteriores de u_{k-i}, z_{k-i}, Q_{k-i}, R_{k-i},y las estimaciones de x_{k-i}.
El filtro consta de dos pasos: una predicción y una corrección.

$$\hat{x}_{k|k-1} = \Phi_k x_{k-1|k-1}$$
$$P_{k|k-1} = \Phi_k P_{k-1|k-1} \Phi_k^T + Q_k$$

Ec 4.1 y 4.2. Predicción

Donde $\hat{x}_{k|k-1}$ es la estimación a priori y $P_{k|k-1}$ es la covarianza del error asociada a dicha estimación.

$$\tilde{y}_k = z_k - H_k \hat{x}_{k|k-1}$$
$$K_k = P_{k|k-1} H_k^T (H_k P_{k|k-1} H_k^T + R_k)^{-1}$$
$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \tilde{y}_k$$
$$P_{k|k} = (I - K_k H_k) P_{k|k-1}$$

Ec 5.1, 5.2, 5.3 y 5.4. Correción

Donde $\tilde{y}_k$ es la actualización de la medida, K_k es la ganancia de Kalman, $P_{k|k}$ es la estimación a posteriori y $P_{k|k}$ es la covarianza del error asociada a la estimación a posteriori. Y siendo Φ_k la matriz de transición de estados que relaciona $x_{k|k-1}$ con $x_{k-1|k-1}$, $x_{k|k-1}$ el valor de estado a priori, $P_{k|k-1}$ la covarianza del error de la estimación a priori, z_k las medidas en el instante *k*, H_k la matriz que relaciona medidas y vector de estado suponiendo ruido de medida nulo, y R_k la matriz de covarianza del ruido de las mediciones.

2.3.4. Triangulación epipolar

Uno de los inconvenientes del uso de cámaras es la pérdida de información debido a la proyección de la realidad, en tres dimensiones, sobre el plano de la imagen, en dos dimensiones. Esto provoca que no sea posible determinar la profundidad de objeto. Una posible solución a este problema es el uso de más de una cámara.
Para la explicación considérese el ejemplo de la figura 2:

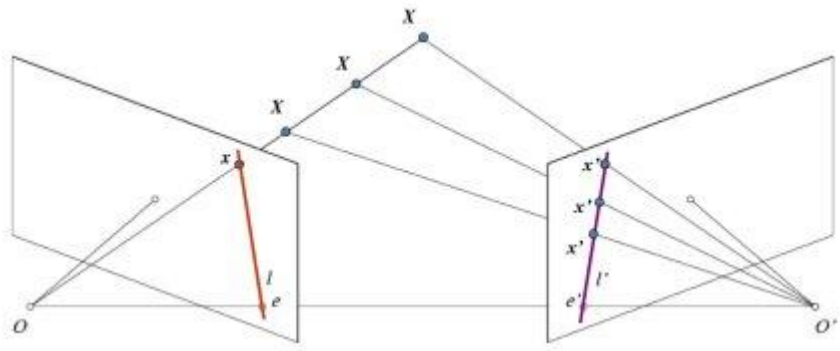


Fig. 2. Geometría epipolar

http://docs.opencv.org/trunk/epipolar.jpg

Nótese que no es posible determinar la posición en el espacio del punto *x* utilizando el plano de la imagen de la izquierda, puesto que todos los puntos de la recta *OX* se proyectan sobre el mismo punto de la imagen. Sin embargo, si se tiene en cuenta la imagen de la derecha, cada una de las posiciones de *x* (proyectadas como *x'*) ocupa una posición diferente. De esta forma se puede triangular el punto correctamente sobre el espacio.

La proyección de los diferentes puntos de *OX* forma una línea en el plano de la derecha, (línea *l'*) denominada epilínea, correspondiente al punto x. Esto significa que el punto *x* en la imagen derecha estará situado sobre esta epilínea, esto se conoce como restricción epipolar. Del mismo modo, todos los puntos de un plano de imagen tendrán su correspondiente epilínea sobre el otro. El plano *XOO'* se conoce como plano epipolar.

Los puntos *O* y *O'* son los centros de las cámaras, y se proyectan sobre el plano de la otra imagen en los puntos *e'* y *e* respectivamente. Este punto se conoce como epipolo, y todas las epilíneas pasan por él.

Para la triangulación también son necesarios otros parámetros, la matriz fundamental y la matriz esencial. La matriz esencial contiene información acerca de la rotación y la traslación de la segunda cámara respecto de la primera, o lo que es lo mismo la posición y rotación de la segunda cámara en coordenadas absolutas tomando como origen la primera.

La matriz fundamental incluye toda la información de la matriz esencial además de los parámetros intrínsecos de las cámaras, lo que permite la conversión de distancias de píxeles a unidades reales.

Con todos estos parámetros es posible triangular un punto en el espacio (figura 3), utilizando las proyecciones sobre la imagen, la correspondencia píxel-distancia real y cálculos trigonométricos. Las distancias *B* y *f* son conocidas de antemano; las distancias *x* y *x'* se miden en píxeles sobre la imagen y su conversión a distancia real también es conocida; por semejanza de triángulos se puede determinar la distancia *z* a la línea base *B*.

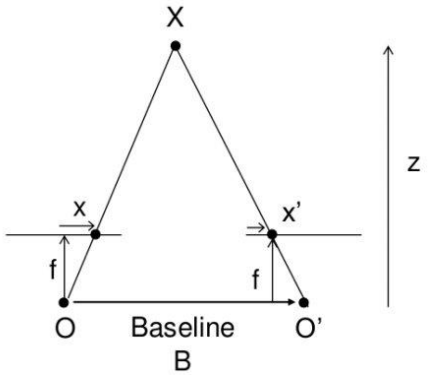


Fig.3. Triangulación epipolar

http://docs.opencv.org/trunk/stereo_depth.jpg

OpenCV utiliza un sistema de coordenadas basado en las imágenes introducidas (figura 4): el eje *x* en dirección horizontal y sentido positivo hacia la derecha, el eje *y* en dirección vertical y sentido positivo hacia abajo y el eje *z* perpendicular a ambos con sentido positivo alejándose de las cámaras formando un sistema dextrógiro. El eje de coordenadas se sitúa sobre el punto *O* de la figura 3 sobre

la cámara situada a la izquierda del par, que corresponde con el punto teórico del obturador en el modelo de cámara de ojo de aguja (*pinhole camera model)*.

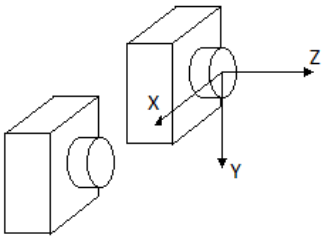


Fig.4. Eje de coordenadas

2.3.5. Calibración de las cámaras

Todos los fundamentos explicados anteriormente permiten convertir las imágenes en parámetros que se pueden manipular matemáticamente para obtener información. Sin embargo la adquisición de estas imágenes conlleva una serie de factores a tener en cuenta, debidos a las propiedades físicas de las cámaras. Para compensar los efectos de estos factores es necesaria una calibración de las cámaras. Existen diferentes métodos de calibración[8], pero el más extendido por su sencillez y buenos resultados es el uso de una plantilla plana[9],[13].

En primer lugar, el modelo lineal básico de cámara se conoce como *pin-hole* y está determinado por la ecuación:

$$q = \lambda K[R\ t]p$$

Ec 6. Modelo pin-hole.

Donde *q* son las coordenadas homogéneas del punto sobre el plano de la imagen, *p* es el punto en el espacio, *R* es la matriz de rotación de la cámara respecto al origen, *t* es su vector de traslación y *K* es la matriz intrínseca de la cámara.

Dadas las características de las lentes de las cámaras, se produce también una distorsión en la imagen. La posición correcta en píxeles del punto *q* sobre la imagen $q_p=(u_p\ v_p)$ está relacionada con la posición observada del mismo $q_d=(u_d\ v_d)$. Esta relación se puede expresar con las ecuaciones:

$$u_d = u_p - \delta_u(u_p, v_p)$$
$$v_d = v_p - \delta_u(u_p, v_p)$$

Ec 7. Distorsión por la lente.

De ellas se observa que la cantidad de error geométrico depende de la posición en la imagen. Según (Brown, 1966), (Faig, 1975), (Stama, *et al*, 1980), p_1, p_2, p_3... modela la distorsión tangencial, la radial se modela con los coeficientes k_1, k_2, k_3... y la de prisma con s_1, s_2, s_3... La distorsión efectiva puede

ser modelada por la suma de los tres tipos anteriores. Descartando los términos de orden superior a 2 se obtiene el modelo no lineal:

$$\delta_u(u_p, v_p) = u_p k_1 r^2 - p_1(3{u_p}^2, + {v_p}^2) + 2p_2 u_p v_p + s_1 r^2$$
$$\delta_v(u_p, v_p) = v_p k_1 r^2 - p_2(3{u_p}^2, + {v_p}^2) + 2p_1 u_p v_p + s_1 r^2$$

Ec 9. Modelo de distorsión.

El método de calibración de Zhang (Zhang 1998, 2000) utiliza las coordenadas de los puntos situados en una plantilla plana 2D, y al contrario que métodos anteriores no requiere gran precisión en el diseño de la plantilla ni una medición exacta de los puntos de la misma. La sensibilidad del algoritmo puede ser mejorada añadiendo más puntos al modelo de tablero de ajedrez.
Se establece un método para la calibración lineal basado en la plantilla plana de Zhang consistente en tomar varias imágenes de la plantilla en diferentes posiciones y orientaciones de la cámara. Con cada una de las imágenes se puede obtener una homografía *H* resolviendo la expresión:

$$\begin{bmatrix} p_i^T & 0 & -u_i q_i^T \\ 0 & p_i^T & -v_i q_i^T \end{bmatrix} h = 0$$
$$A \cdot h = 0$$

Ec 10. Homografía.

Calculando *n* homografías a partir de *n* imágenes de la plantilla se obtiene un sistema de ecuaciones de la forma $V \cdot b = 0$, donde *V* es una matriz de dimensiones $2n \times 6$, formada con los elementos de las homografías. Para $n \geq 3$ se obtiene una solución general de *b* definida con un factor de escala. La solución del vector *b* es el vector propio de $V^T \cdot V$ asociado al valor propio más pequeño. A partir del vector estimado *b* se obtienen los parámetros extrínsecos de la cámara. Una vez conocidos los parámetros intrínsecos, se calculan los extrínsecos a partir de cada homografía.
Si se realiza una estimación no lineal de los parámetros, se minimiza un error residual definido por la expresión $e = b^{\#} - f(a)$, donde a es el vector de parámetros que deben satisfacer las medidas $b^{\#}$ (conteniendo ruidos) con la función lineal *f*. En la calibración de la cámara, *e* representa la distancia geométrica entre los puntos medidos en la imagen y las coordenadas de los puntos proyectados con la plantilla de calibración. La función *f* expresa la relación entre coordenadas de los puntos en la imagen y la plantilla y habitualmente se utilizan los parámetros del modelo lineal que se calcula directamente.(Ricolfe, Sánchez, 2008)

Este método ha sido perfeccionado a lo largo de los últimos años[10],[11],[12], dando lugar a un sistema muy robusto y altamente popular.

2.4. Motivación

El presente trabajo pretende evaluar dichas herramientas en cuanto a su aplicabilidad a videos de alta velocidad. Se pretende sentar una base empírica sobre la efectividad de los métodos de optimización para aplicaciones de control robótico en los que la velocidad es un factor clave.

2.5. Justificación

Se evaluarán diversos métodos de optimización para el procesado de imágenes en función principalmente de cómo mejoran la velocidad de procesado, aunque también se tendrán en cuenta la calidad de las medidas obtenidas. Se pretende establecer una base experimental para futuras aplicaciones.

3. Normativa

Cada uno de los elementos empleados para el desarrollo del presente trabajo se regula por la normativa correspondiente a su naturaleza.

Cámaras *Mikrotron EoSens CL*, modelo *MC1363*:

Están sujetas a las regulaciones de las Directivas del Consejo de la UE 2004/108/EG en el ámbito de la consistencia electromagnética EN 61000-6-3 de compatibilidad electromagnética y EN 61000-6-1 sobre inmunidad.

El software de control *Mikrotron Control Tool* es descargable de la página del fabricante bajo licencia.

Tarjeta *Micro Enable IV AD4-CL* de *Silicon Software*:

El hardware de adquisición de imagen viene regulado en conformidad a las directivas de la CE.

El software de control está incluido con el hardware bajo licencia de *Silicon Software*. Este software incluye código creado por otros autores y utilizado bajo sus respectivas licencias.

Librerías de software de OpenCV:

Las librerías de código abierto se utilizan bajo su licencia correspondiente.

4. Ámbito de aplicación

4.1. Cámaras

Para la aplicación se han empleado dos cámaras iguales modelo Mikrotron MC 1363. Este modelo permite tomar imágenes en color (filtro Bayer) de hasta 1280x1024 y hasta una velocidad de 1594 fps, dependiendo del tamaño de imagen. La cámaras disponen de entrada de sincronismo para asegurar la simultaneidad de los pares de imágenes. Además incluyen gran cantidad de parámetros internos ajustables que permiten reducir la necesidad de procesado de la imagen. Disponen de un conector para el control y transmisión de datos estándar tipo *Camera Link*.

Para la aplicación se han utilizado los siguientes parámetros, se introducen mediante el programa controlador *Mikrotron Control Tool*:

- Tamaño de la imagen: 512 x 512 píxeles, centrada.
- Salida: 2 x 8bits.
- Ganancia: 1.5.
- Balance de blancos: 160 (+32).
- Disparo: entrada de sincronismo con temporizador, exposición de 2500µs.
- Triple pendiente activada: segunda pendiente en 75 y tercera en 95.

La múltiple pendiente permite corregir las zonas excesivamente brillantes, reduciendo su tiempo de exposición en un 75%, si siguen teniendo un brillo excesivo se reduce nuevamente su exposición al 95%.

Las cámaras se alimentan mediante una fuente externa de 24V 0.6A. Están atornilladas sobre un marco metálico paralelamente, separadas 180mm (figura 5). Disponen de un objetivo que regula el enfoque y la apertura del obturador.

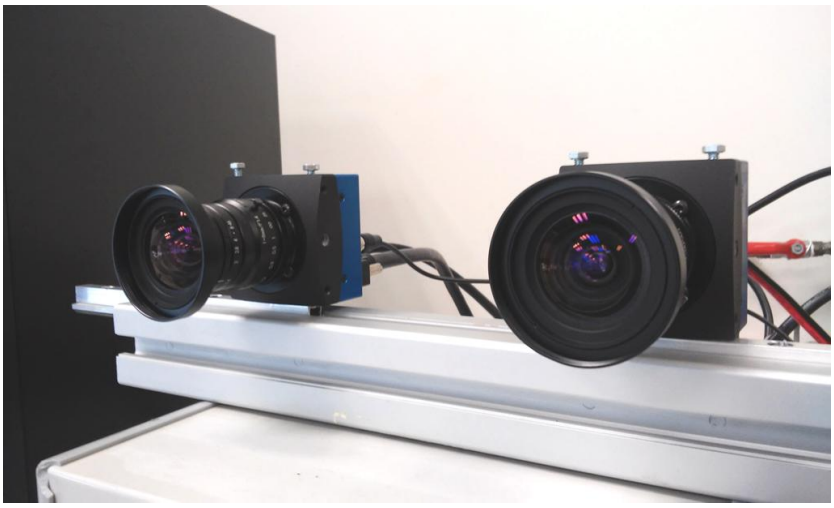


Fig. 5. Montaje de las cámaras.

4.2. Tarjeta

Se ha empleado la tarjeta Micro Enable IV AD4-CL de Silicon Software. Dispone de dos entradas *Camera Link* que permiten la recepción simultánea de dos series de imágenes. Dispone de 256MB de memoria RAM interna y una capacidad máxima de transmisión de 850MB/s. Para controlar el sincronismo de las imágenes se ha utilizado la fuente interna de disparo de hasta 100Hz a máxima velocidad.

La tarjeta dispone de un procesador interno que permite preprocesar los datos recibidos. Se emplea la reconstrucción del color mediante el filtro Bayer de la cámara (bloques según la estructura verde-azul-rojo-verde) y se realiza un corrección de color aumentando la saturación de los canales azul y rojo en un 10%. Los demás parámetros se configuran en correspondencia con las cámaras.

Fig. 6. Micro Enable IV AD4-CL.

https://silicon.software/wp-content/uploads/img-prd-fg-me4d4cl17.jpg

La tarjeta dispone de un puerto para el conexionado de una fuente externa que proporcione una señal de sincronismo. La principal razón por la que se incluye es que en la mayoría de aplicaciones se utilizan velocidades mucho mayores a las que es capaz de generar el reloj interno de la tarjeta. En esta aplicación no obstante, no se espera que el sistema sea capaz de procesar las imágenes a velocidades superiores a las de la fuente de disparo, por lo que para reducir la complejidad del montaje se escoge no montar un sistema de disparo externa.

También es notable destacar el hecho de que la tarjeta va instalada en el ordenador, por lo que en última instancia, la velocidad real a la que la tarjeta es capaz de adquirir imágenes viene limitada por la velocidad a la que el bus interno de datos es capaz de transmitirlos. Para maximizar la velocidad de comunicación, la tarjeta utiliza un puerto PCI express de cuatro canales. Se debe tener cuidado a la hora de configurar el sistema, de asegurarse de que otros dispositivos, como habitualmente la tarjeta gráfica, no estén ocupando los puertos.

4.3. Programa

Si bien la tarjeta dispone de un programa específico para la captura de imagen y vídeo, no permite el procesado de las imágenes, por lo que se ha empleado la API de desarrollo también incluida por el fabricante. La API *Silicon Software SDK* permite la configuración de la tarjeta y otras funcionalidades del programa de captura a través de código en lenguaje C++. Para la captura se ha utilizado la *Applet* de preconfiguración *Acq_DualBaseAreaBayer8*, que permite la captura en paralelo de dos series de imágenes en color por filtro de Bayer con una profundidad de color de 8 bits.

La entrada de imágenes consta de dos procesos en paralelo. Por un lado la tarjeta recibe las imágenes desde las cámaras a una frecuencia constante determinada por la velocidad de disparo, éstas se almacenan en un buffer de memoria en anillo junto con su número de fotograma. El buffer dispone de *n* fotogramas (en este caso 4) para poder acceder a ellos sin problemas de lectura y escritura simultáneas por parte de procesos distintos. Por otra parte, el algoritmo de procesado accede al último par de fotogramas disponibles y crea una copia para realizar las operaciones que necesite. Esta copia es necesaria puesto que el tiempo de procesado no es conocido y es posible que la tarjeta sobrescriba la imagen que se está procesando actualmente si tarda demasiado. Cuando se completa el procesado se toma una nueva copia del actual último par disponible, repitiendo el proceso.

Para la detección de objetos se ha optado por seguir una esfera de color rojo intenso de 50mm de diámetro. Esta decisión se basa en dos factores: uno es simplificar la segmentación mediante color, tomando uno de los colores básicos de la imagen (rojo, verde o azul); aunque los tonos verdes o azules permiten mejor separación de la forma a seguir de los tonos de la piel humana, la diferencia de intensidad es suficiente para distinguir la esfera incluso sosteniéndola en la mano. El otro factor es la forma, una esfera, que invariablemente de su posición u orientación siempre se proyecta en una imagen como un círculo, permitiendo una detección más sencilla y sin necesidad de casuística.

Fig 7. Objeto a seguir.

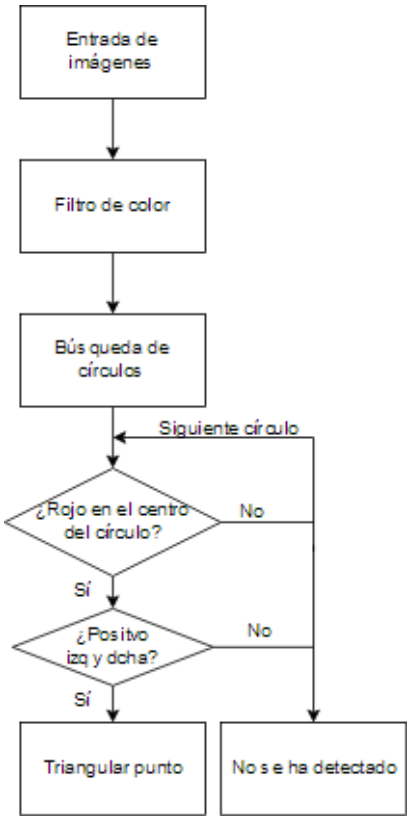


Diagrama 1. Diagrama de bloques del algoritmo de detección.

El proceso de detección consta de tres pasos: en primer lugar se separa el color y se extraen los tonos rojos de la imagen, restando al canal rojo los canales verde y azul multiplicados por un coeficiente. De este modo se obtiene una imagen de un solo canal en la que los tonos rojos aparecen brillantes, y otros tonos (incluyendo blancos y grises, que tienen un alto valor de rojo pero también de verde y azul) aparecen oscuros.

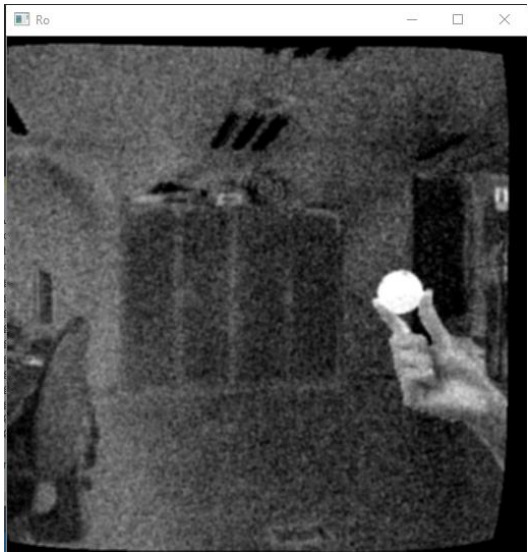


Fig. 8. Filtrado de color. La esfera aparece como un círculo brillante.

El segundo paso consiste en la aplicación del algoritmo de detección de círculos por gradiente de Hough. OpenCV dispone de una función para detectar círculos en una imagen, utilizando dicha función se obtienen las zonas donde es más probable que haya un círculo, y,de los círculos obtenidos se escoge sólo aquel que tenga un alto nivel de rojo en su centro (alto brillo en la imagen filtrada).

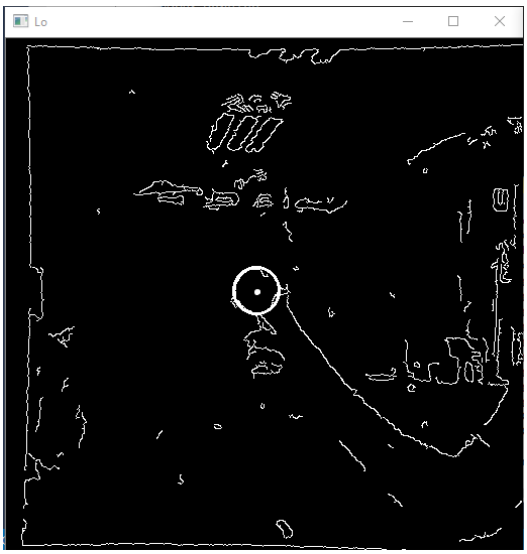


Fig. 9. Resultados de la detección de círculos sobre bordes detectados.

El tercer paso consiste en la triangulación en el espacio de la posición de la esfera mediante su posición proyectada en las dos imágenes. Para poder llevar a cabo este último paso es necesario un sistema de matrices que permitan calcular la correspondencia de puntos y se obtienen mediante una calibración previa.

El sistema utiliza la matriz fundamental para reconstruir los puntos además de una corrección para ajustar la correspondencia de puntos en los valores medidos.

5. Factibilidad e impactos

El proyecto está basado en los recursos tecnológicos disponibles, bien por los recursos materiales (ordenador, cámaras) presentes en el laboratorio, o bien por el software de acceso libre (OpenCV). Se ha empleado el lenguaje C++, altamente extendido, y con gran cantidad de recursos adicionales disponibles en la red. Por lo tanto, desde un punto de vista tecnológico, el proyecto es claramente asequible.

Dado el carácter de investigación del proyecto, tiene escaso retorno a nivel económico, aunque puede utilizarse para ayudar a mejorar la productividad de posibles futuros proyectos.

Del mismo modo, el estudio tiene repercusiones muy limitadas más allá del ámbito del desarrollo de software de seguimiento de objetos. Dentro de este campo, el proyecto pretende servir como referencia en la toma de decisiones en la elección de las soluciones más adecuadas en el desarrollo de nuevos proyectos.

6. Condiciones de trabajo

Para llevar a cabo este estudio de análisis se ha elaborado un programa base de detección de objetos mediante la detección de tanto color como forma. Este método aporta mayor robustez a la detección, pero también aumenta la cantidad de procesado necesario reduciendo por consiguiente la velocidad efectiva a la que se puede procesar la imagen.

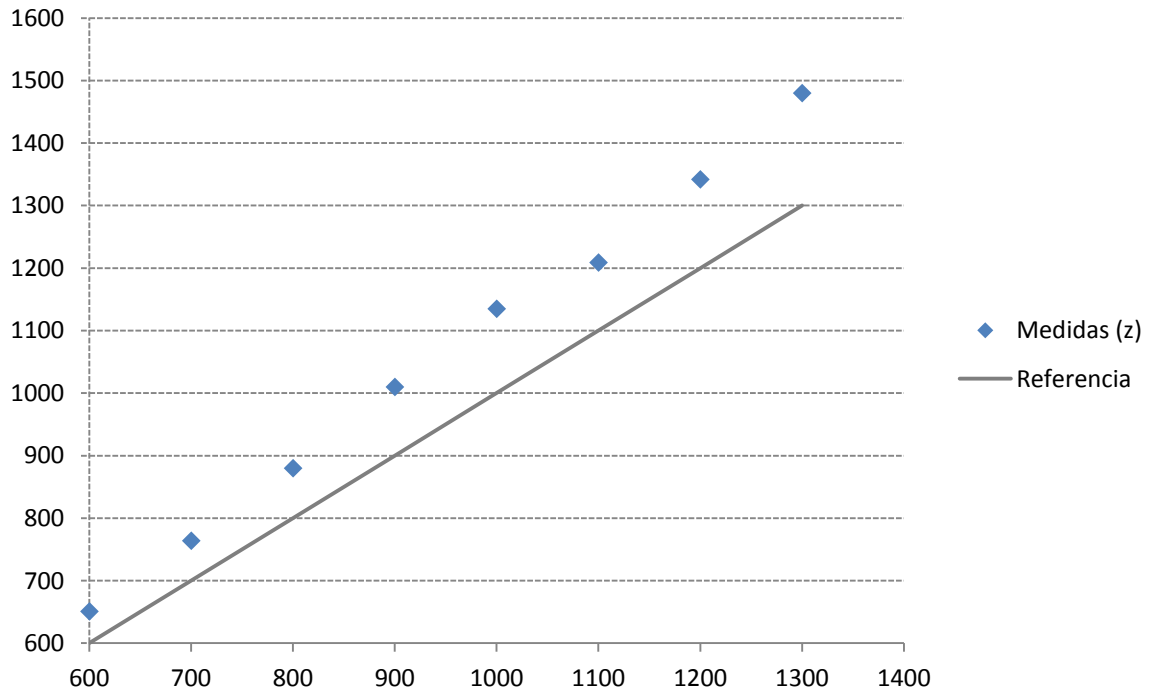
La detección por color es más simple y requiere menor tiempo de cálculo, pero es mucho más propensa a dar resultados de menor calidad. La triangulación del objeto se realiza utilizando como punto el centroide del *blob*, por lo que es más proclive a desplazarse por distorsiones en la forma detectada a causa de luces, sombras u objetos de color similar cercanos al objeto a detectar (en este caso particular el color de la piel humana cerca de la esfera roja al sostenerla en la mano).

La detección por forma se introduce para dar mayor robustez al sistema, y dar apoyo para futuras aplicaciones en las que no sea posible (o conveniente) la detección por los métodos más simples. Este método de detección requiere tiempos de cálculo significativamente mayores y depende en gran medida del tamaño de la imagen a procesar.

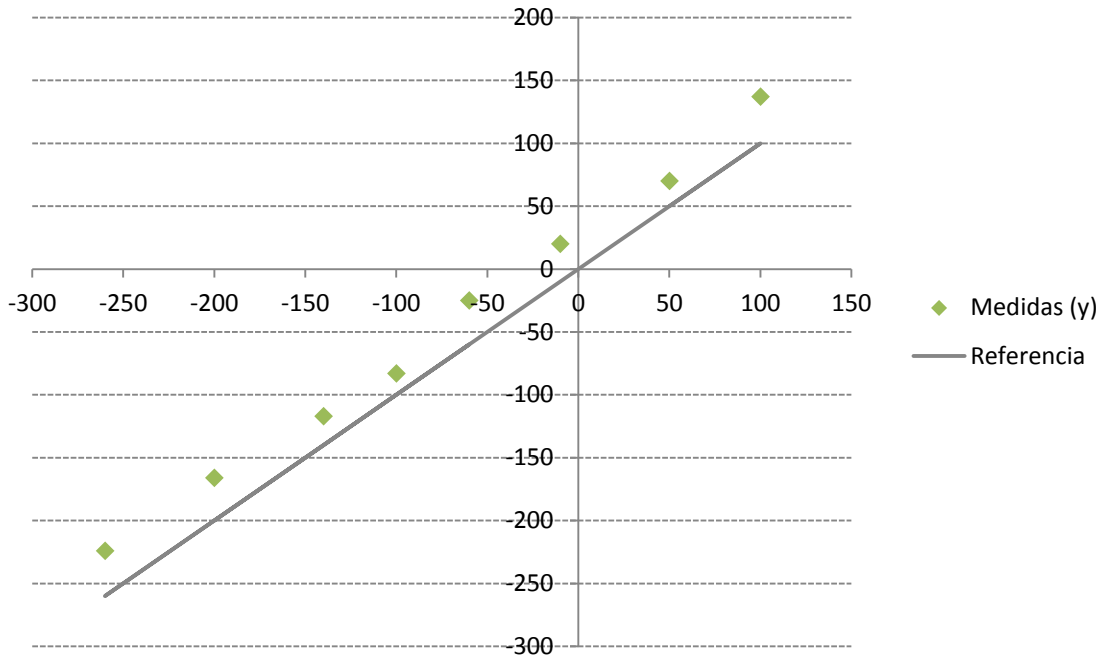
A continuación se muestran los resultados en los diferentes ensayos:

6.1. Precisión

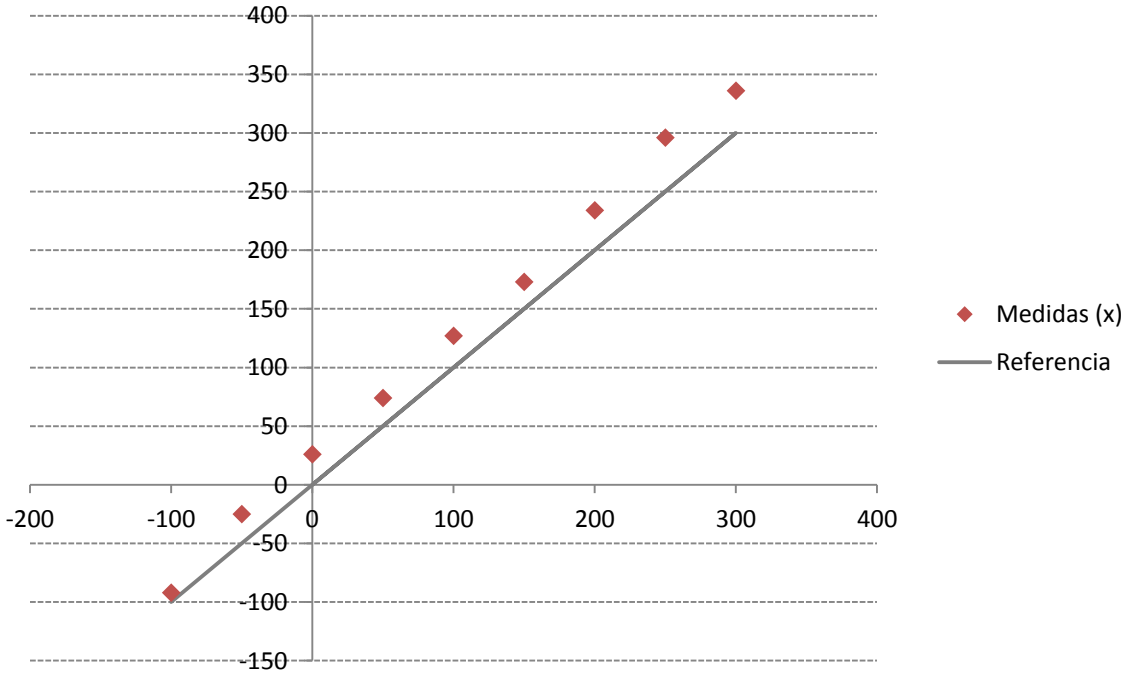
Los resultados a nivel de precisión a la hora de determinar la posición del objeto son altamente consistentes puesto que no muestran ninguna variación sensible dependiendo de la versión programa utilizada. Por lo tanto no se considera que los diferentes métodos de optimización tengan efectos significativos sobre la precisión.



Gráfica 1. Distancia medida en el eje z, respecto a distancia real, para x, y constantes (x = 0, y = 200)



Gráfica 2. Distancia medida en el eje y, respecto a distancia real, para x, z constantes (x = 0, z = 880)



Gráfica 3. Distancia medida en el eje x, respecto a distancia real, para z, y constantes (z = 860, y = 200)

Los resultados obtenidos presentan un error sistemático consistente, debido probablemente a la calibración. Para obtener las matrices de calibración se han empleado dos pares de imágenes de un patrón de tablero de ajedrez de 8x6 cuadros, con lado de 108mm. Para obtener mejores resultados podría ser conveniente utilizar más imágenes, mostrando el patrón en más posiciones y orientaciones en el plano de la imagen. No obstante, puesto que el sistema utiliza correspondencia de puntos entre los pares de imágenes, hay una zona en cada imagen del par en la que el punto correspondiente está fuera de la otra imagen; en esta zona es difícil obtener una calibración correcta.

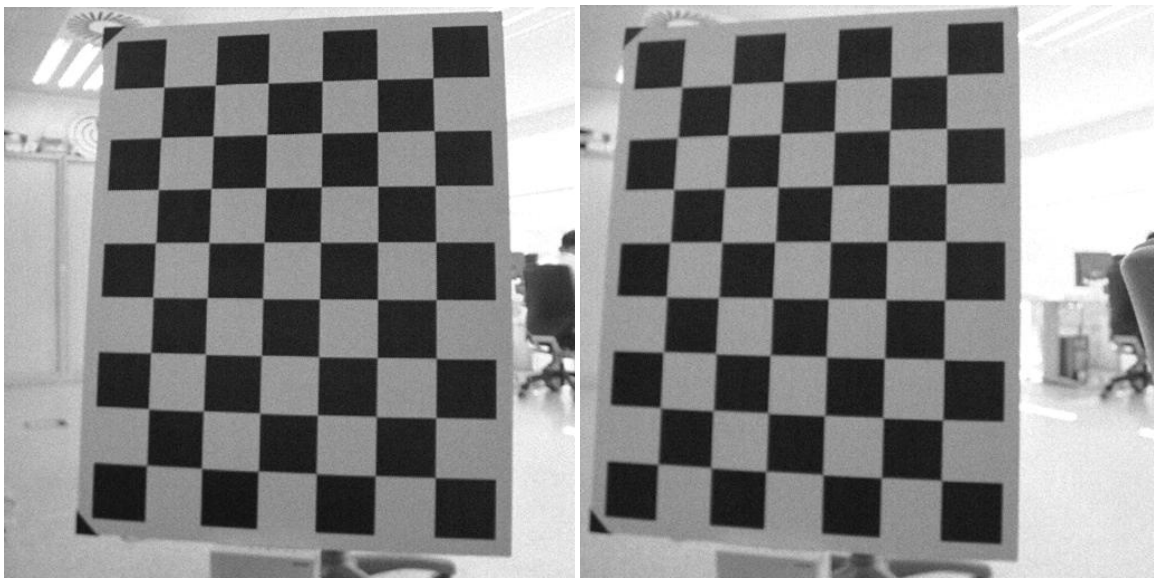
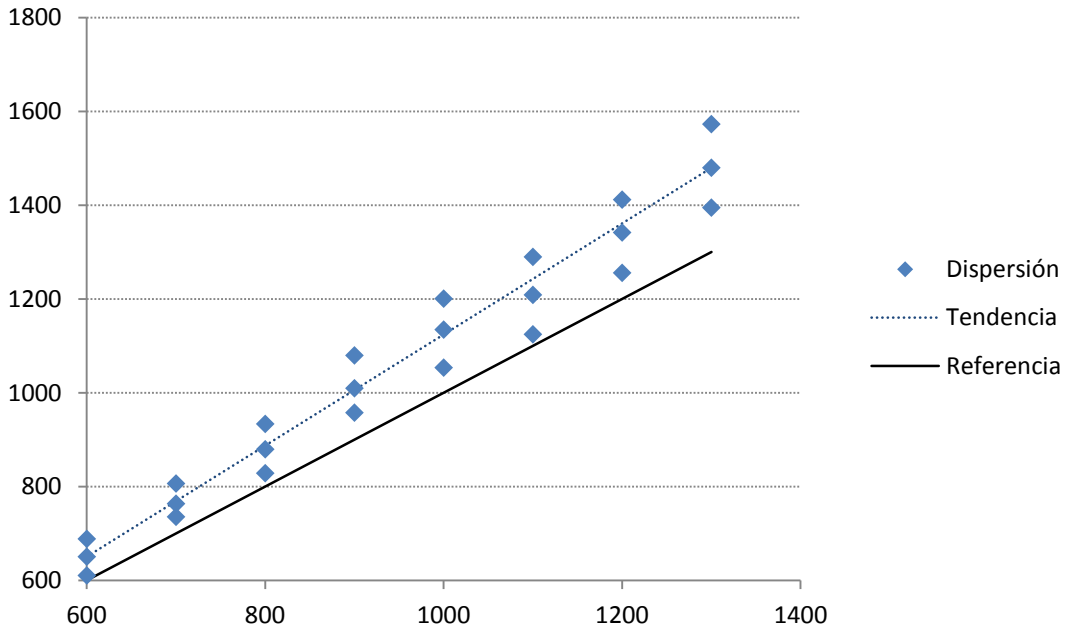


Fig. 10. Pareja de imágenes de calibración.

Los valores obtenidos de la calibración se introducen en el programa de seguimiento a través de un fichero externo *"StereoCalibration.yml"*. Este fichero contiene las matrices de calibración obtenidas de la función de OpenCV *StereoCalibrate*.

Otra característica a tener en cuenta de los resultados obtenidos es la oscilación de los valores. Para un valor fijo de posición (correspondiente a un objeto estático) se obtienen resultados con una cierta oscilación, esto se debe al error intrínseco del método de detección de formas. El ajuste de un círculo en los contornos detectados provoca un ligero desplazamiento del centro, que se usa como punto de referencia. Una forma de corregir este problema es el uso de un filtro de media a la hora de mostrar los resultados. No se ha implementado en el sistema debido a que ralentiza la respuesta ante movimientos rápidos y afectaría negativamente al otro factor de estudio, la velocidad a la que es capaz de proporcionar medidas.



Gráfica 4. Dispersión de valores medidos en el eje z, línea de tendencia y referencia de distancia real.

(Para x, y constantes: x = 0, y = 200)

El sistema es capaz de detectar el objeto en un volumen en forma aproximada de tronco de pirámide rectangular (debido al área de visión) desde un área de 300 x 400mm de lado a 600mm de las cámaras hasta un área de 1.5 x 2m de lado a 1300mm de las cámaras. La zona óptima de trabajo comprende las distancias entre 600 y 1000mm.

6.2. Velocidad

El parámetro al que más atención se ha prestado en este estudio es la velocidad de procesado del vídeo de entrada. Los vídeos de alta velocidad suelen tener una velocidad de varios miles de fotogramas por segundo. Es poco realista pretender que un PC convencional sea capaz de procesar varios miles de pares de imágenes por segundo; no obstante, los métodos de seguimiento de objetos convencionales suelen operar en rangos comprendidos entre los 10 y los 30 fotogramas por segundo. Este estudio pretende analizar los efectos de diversos métodos de optimización para mejorar estos valores. Para medir la velocidad a la que el sistema es capaz de procesar las imágenes y obtener un resultado se ha empleado el contador de fotogramas interno de la tarjeta: cada valor obtenido se acompaña del número de fotograma del cual se ha calculado y sabiendo que se ha establecido la velocidad de disparo de las cámaras a 100 fotogramas por segundo, contando cuantos valores se obtienen por cada 100 pares de imágenes se obtiene una buena aproximación de la velocidad de procesado.

6.2.1. Método off-line

El principal problema del procesado de vídeo de alta velocidad es el escaso tiempo entre fotogramas para su procesado. La solución más sencilla es el procesado a posteriori, que consiste en grabar primero el vídeo y posteriormente procesar fotograma a fotograma. De este modo se elimina completamente el problema de la velocidad, el procesador dispone de todo el tiempo que necesite (dentro de unos límites razonables) para tratar cada imagen. Este método tiene una aplicabilidad muy limitada y a efectos prácticos el hecho de que el vídeo sea de alta velocidad no tiene ningún efecto sobre el procesado. Por este motivo, no se profundizará más en él.

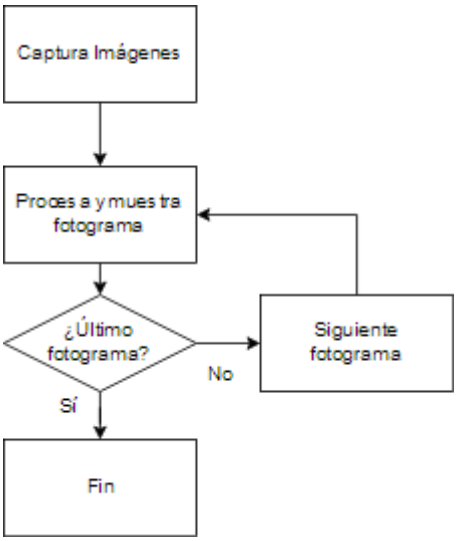


Diagrama 2. Diagrama de bloques del algoritmo off-line.

6.2.2. Métodos on-line

En la mayoría de aplicaciones se suele requerir el procesado del vídeo en tiempo real, para poder emplearlo como entrada de información para un sistema de control.

6.2.2.1. Algoritmo básico

Como punto de referencia se ha utilizado un método básico de detección de objetos. El algoritmo busca el objeto a seguir en las imágenes y si obtiene un resultado positivo en ambas, triangula su posición.

Este método presenta la velocidad más estable, aunque también la más lenta, a un promedio de 15.4 pares de imágenes por segundo.

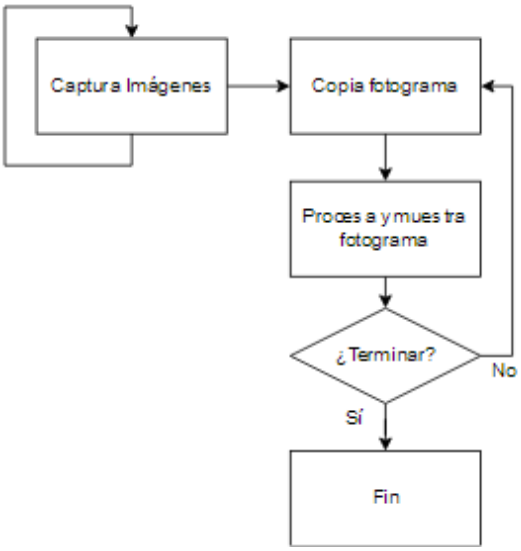


Diagrama 3. Diagrama de bloques del algoritmo on-line básico.

Los siguientes algoritmos se basan en el mismo procedimiento, pero incluyen sistemas para mejorar la velocidad de los resultados.

6.2.2.2. Modelo de predicción

El principal determinante de la velocidad de procesado es el tamaño de la imagen, por lo tanto el principal factor a reducir es el área de búsqueda. Esto introduce un nuevo problema: la predicción de posiciones futuras del objeto en la imagen para determinar el área en la que es más probable que se encuentre el objeto . Por este motivo se debe determinar un modelo de comportamiento del objeto a seguir. En este caso se ha escogido utilizar el modelo de sólido libre, afectado únicamente por la aceleración de la gravedad. Este modelo es muy limitado puesto que no tiene en cuenta ningún otro tipo de aceleración, pero dado que en los ensayos, el movimiento de la esfera viene determinado por los movimientos de la mano al sostenerla, es completamente arbitrario y muy complejo de modelar.

Para los ensayos se ha escogido buscar la esfera en un área de 120x120 píxeles, lo que supone una reducción de la superficie de búsqueda del 94.5%.

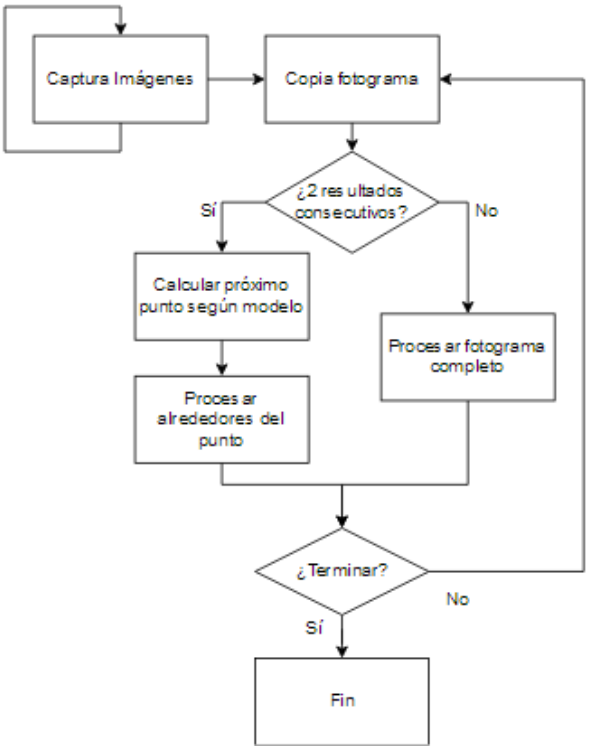


Diagrama 4. Diagrama de bloques del algoritmo on-line con modelo de predicción.

Para la estimación de la posición se ha empleado un modelo discretizado, asumiendo aceleración constante de 9800mm/s^2 hacia abajo (sentido positivo del eje *y*). Se ha tenido en cuenta la variabilidad de tiempos de procesado, midiendo el tiempo transcurrido entre la entrada de una imagen a procesar (instante *n*) y la siguiente imagen que se procesa, que puede ser o no la siguiente imagen capturada (instante *n+1*). Los puntos en el tiempo se marcan a la entrada de la imagen, antes de procesar, por lo que el instante *n* es la imagen ya procesada, de la que se dispone un resultado, y el instante *n+1* es la nueva imagen, de la que ya se tiene instante de entrada t_{n+1} y se pretende calcular su posición Q_{n+1}. El modelo se aplica sobre la posición triangulada en el espacio y se reproyecta sobre el plano de las imágenes.

El modelo se obtiene por doble integración de la aceleración $\ddot{Q}$ (segunda derivada de la posición, con valor *g = 9800* en el eje *y,* y 0 en *x*, *z*), suponiendo la velocidad constante en el intervalo temporal (t_{n-1}, t_n) de valor $\frac{\Delta Q}{\Delta t} = \frac{Q_n - Q_{n-1}}{t_n - t_{n-1}}$.

$$Q_{n+1} = \begin{bmatrix} X_n + \dfrac{X_n - X_{n-1}}{t_n - t_{n-1}} \cdot [t_{n+1} - t_n] \\ Y_n + \dfrac{Y_n - Y_{n-1}}{t_n - t_{n-1}} \cdot [t_{n+1} - t_n] + \dfrac{g}{2} \cdot [t_{n+1} - t_n]^2 \\ Z_n + \dfrac{Z_n - Z_{n-1}}{t_n - t_{n-1}} \cdot [t_{n+1} - t_n] \end{bmatrix}$$

Eq. 6. Modelo de sólido libre

Es importante destacar que, debido a la forma en la que OpenCV trata las imágenes en color, cuando se trabaja sobre ellas es necesario invertir los ejes *x* e *y*. Esto se manifiesta en el código tanto a la hora de determinar los centros de los círculos detectados como a la hora de reproyectar el punto estimado en el espacio sobre las imágenes.

Los resultados de la introducción de este método son claramente positivos, pese al aumento en los cálculos necesarios, cuando el objeto a seguir se mantiene en la zona de predicción, la velocidad de imagen sube hasta un promedio de 74.2 pares de imágenes por segundo.

También presenta algunos inconvenientes: para poder utilizar el modelo se necesitan al menos dos valores triangulados consecutivos (para poder calcular la velocidad), si se pierde el objeto en algún momento debido a que se sale de la zona de detección o a ocultaciones parciales o totales, o por una aceleración inesperada y brusca se aleja del modelo; el sistema debe volver a procesar la imagen completa, lo que provoca que la velocidad vuelva a caer a cerca de 15 pares de imágenes por segundo hasta que se consiguen dos resultados positivos consecutivos.

En situaciones en las que el modelo está muy alejado de la realidad (gran cantidad de aceleraciones no contempladas) el uso de este sistema es contraproducente debido a que el modelo toma gran cantidad de "fotogramas vacíos" en los que el objeto no se encuentra en la zona esperada, por lo que no puede ser detectado, y sí lo habría sido de haber buscado en la imagen completa.

En otras aplicaciones en las que existan aceleraciones distintas a la gravedad pero conocidas y modelables, pueden ser tenidas en cuenta, puesto que el aumento en cantidad de cálculo para el modelo es mucho menor que las necesidades para la detección de forma en imágenes significativamente más grandes.

6.2.2.3. Predicción estática

El algoritmo anterior funciona a un velocidad relativamente alta mientras el objeto se ajuste al modelo y permanezca detectable, sin embargo, en cuanto se pierde el objeto, la velocidad cae drásticamente. Esto hace que sea muy sensible a perturbaciones de corta duración como ocultaciones totales o parciales, destellos o sombras. Para mitigar estos efectos, se ha añadido al algoritmo un sistema que supone el objeto sobre la última ubicación predicha si no es detectado. Se supone el objeto en un área cercana a la última predicción, y se acota el área de búsqueda en las inmediaciones de este punto durante *n* fotogramas (10 en el caso del ensayo). Puesto que la zona de búsqueda se determina por una suposición, aparece una incertidumbre en la posición que aumenta con cada resultado negativo. Teniendo en cuenta dicha incertidumbre, en cada nueva búsqueda se aumenta el área de escaneo en 10 píxeles en cada dirección, volviendo a las dimensiones mínimas originales en cuanto se obtiene un resultado positivo. Si después de *n* iteraciones aún no se ha detectado el objeto, se considera que la incertidumbre es demasiado elevada (también el área de búsqueda que habrá alcanzado los 320x320 píxeles) y se opta por buscar en la imagen completa.

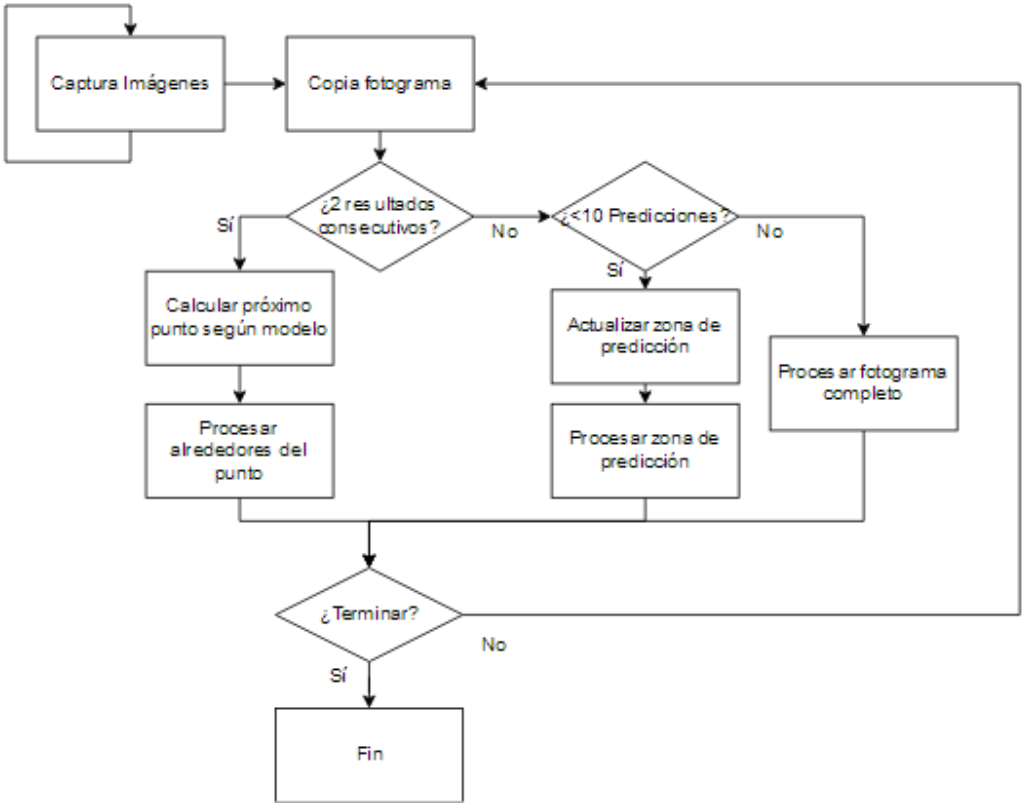


Diagrama 5. Diagrama de bloques del algoritmo con predicción estática.

El uso de este sistema no parece afectar significativamente a la velocidad de procesado cuando el objeto se ajusta al modelo de predicción. Cuando se pierde el objeto a seguir la velocidad se reduce progresivamente hasta alcanzar los 15 pares de fotogramas por segundo cuando se busca en la

imagen completa. Este método resulta útil para aplicaciones en las que pueda haber ocultaciones que duren pocos milisegundos (1-3 fotogramas) causando caídas mucho menores en la velocidad de procesado.

6.2.2.4. Predicción con filtro de Kalman

En el caso anterior se supone que el objeto permanece estático cuando se pierde. Es una suposición bastante alejada de la realidad, puesto que el principal motivo para usar vídeo de alta velocidad es para la detección de objetos de movimiento rápido. Una alternativa es el uso de un filtro de Kalman.

El filtro de Kalman se emplea para la predicción y modelado de trayectorias teniendo en cuenta las posiciones detectadas y un posible error.

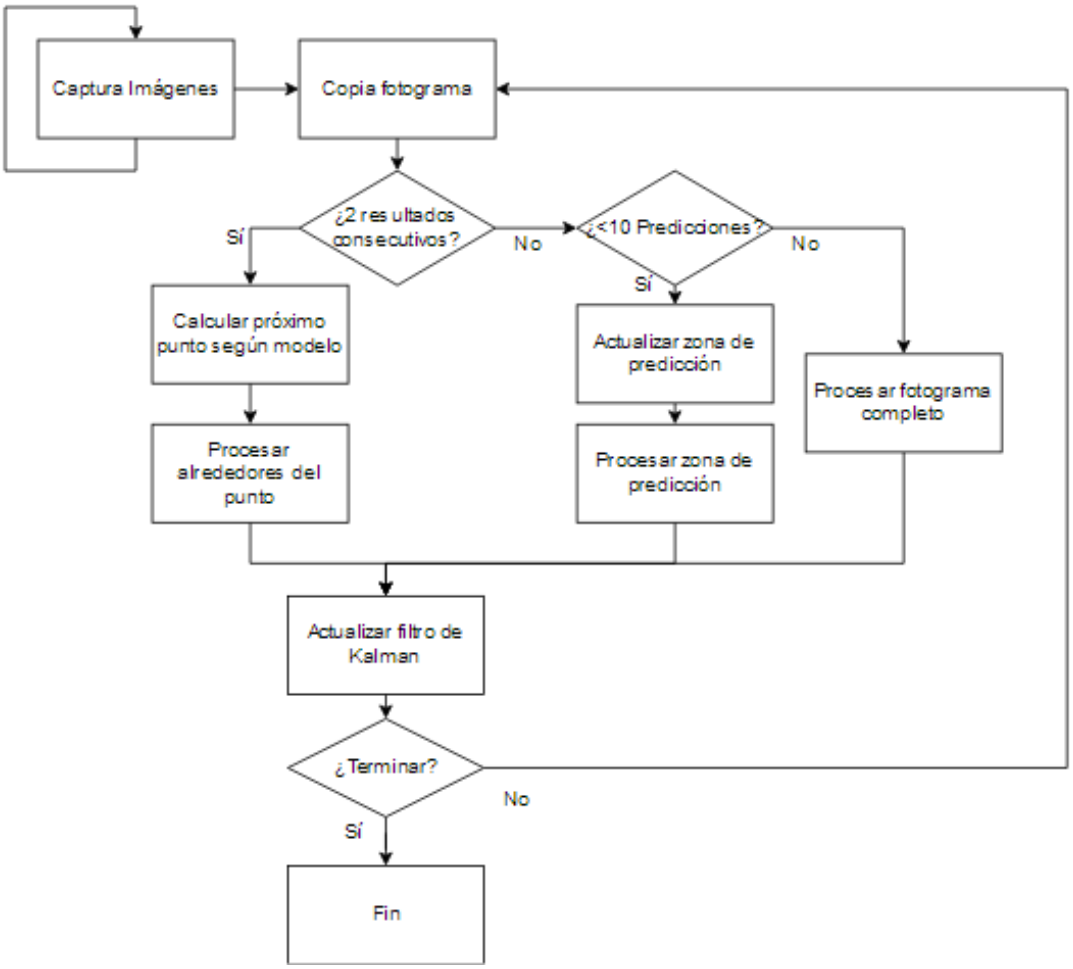


Diagrama 6. Diagrama de bloques del algoritmo con predicción por filtro de Kalman.

Para el algoritmo se ha añadido paralelamente un filtro de Kalman (también incluido en las librerías de OpenCV) que elabora un modelo teniendo en cuenta posición, velocidad y aceleración a través de los puntos detectados. El sistema utiliza realmente dos filtros, uno para cada fotograma del par,

modelando la trayectoria del centro de la esfera en el plano de la imagen. Cuando no se detecta el objeto en la imagen, el algoritmo utiliza la predicción del filtro de Kalman para determinar la posición en el siguiente fotograma. Se ha empleado una fórmula similar al caso anterior aumentando el área de búsqueda en cada fotograma consecutivo en el que se usan predicciones.

Este algoritmo afecta ligeramente a la velocidad de procesado reduciéndola a un promedio de 71.6 pares de fotogramas por segundo. El sistema resulta efectivo en casos en los que se produzcan ocultaciones largas manteniendo aceleraciones constantes. Sin embargo, en los ensayos, la principal causa de la pérdida del objeto son las aceleraciones no modeladas, que causan cambios bruscos en la dirección del objeto. Esto provoca que el algoritmo tienda a seguir buscando la esfera manteniendo la trayectoria anterior cuando se mueve en una dirección completamente diferente. Esto crea situaciones en las que se obtienen posiciones predichas muy alejadas de la realidad y tienda a realizar las 10 iteraciones sin haber encontrado el objeto resultando menos efectivo que los métodos anteriores, puesto que los "fotogramas vacíos" ocupan más tiempo que el procesado incluso de la imagen completa.

Es destacable que debido al funcionamiento interno de las librerías de OpenCV, en todas las versiones se ha dejado una pausa de 1ms en cada ciclo que permite la actualización de las imágenes mostradas. Tanto esta pausa como el tiempo de procesado interno para realizar funciones de dibujado cuya función es puramente de supervisión del correcto funcionamiento del programa (como el dibujado del contorno detectado de la esfera o las áreas que se analizan en cada fotograma) afectan negativamente a la velocidad; aunque se han considerado despreciables respecto a la función de detección de forma, con carga de cálculo mucho mayor. Se puede mejorar el tiempo de computación reduciendo o incluso eliminando estas funciones, o limitando su activación cada *n* fotogramas.

7. Conclusiones

A lo largo de este estudio se ha expresado la imposibilidad, debido a las limitaciones tecnológicas, de alcanzar velocidades de procesado suficientemente altas como para poder operar con vídeo real de alta velocidad, del orden de miles de fotogramas por segundo, en tiempo real. No obstante, se puede apreciar claramente la efectividad de los métodos de optimización, en especial la reducción del área de búsqueda, que han podido aumentar la velocidad de procesado en alrededor de un 500%, y superando con creces la velocidad máxima a la que pueden operar muchas cámaras convencionales.

En cuanto a la precisión de las medidas, si bien tienen una calidad aceptable, el error sistemático parece corregible. No obstante, se ha decidido no emplear demasiado tiempo en un procedimiento para conseguir una calibración exhaustiva, puesto que el desarrollo de métodos de mejora de la calibración se aleja del objetivo principal de este estudio. En este análisis se pretendía estudiar los efectos de los métodos de optimización sobre la precisión de las medidas y, como se ha podido observar, el hecho de tomar las medidas no de la imagen completa sino de una parte y añadir un offset no tiene ningún efecto significativo.

Los métodos propuestos en este trabajo son solo algunas de las posibilidades que se pueden aplicar a casos en los que puede ser necesario utilizar vídeo de alta velocidad para el control de un sistema robotizado. Estas soluciones tienen un carácter general y pueden aplicarse a la mayoría de situaciones. Existen por supuesto, muchas otras alternativas de optimización que pueden resultar incluso más efectivas que las descritas, pero que quedan limitadas a casos particulares. De hecho, la última técnica mostrada, el filtro de Kalman, ha demostrado ser poco eficiente en los ensayos, debido a que en este tipo de situaciones, cuando no se detecta el objeto, suele ser debido a cambios bruscos de dirección, que se alejan del modelo de problema que resuelve el filtro.

Este estudio pretende ser un punto de partida para futuras aplicaciones de control de sistemas robotizados, permitiendo ayudar en la toma de decisiones a la hora de mejorar el rendimiento de un proceso. Por lo tanto, podría ser conveniente más adelante, estudiar cómo afectan tanto estos métodos de optimización como otros en futuras aplicaciones.

Por supuesto, en los trabajos de índole académica es siempre destacable su carácter didáctico, y este proyecto ha permitido ampliar y reforzar los conocimientos en el campo de la visión artificial, que es un campo todavía en proceso de desarrollo y con grandes expectativas en el futuro tanto dentro del ámbito de la robótica y la automática como en general dentro de la ingeniería industrial.

8. Bibliografía

[1]- Sánchez, A. J., & Martinez, J. M. (2000). Robot-arm pick and place behavior programming system using visual perception. In Pattern Recognition, 2000. Proceedings. 15th International Conference on (Vol. 4, pp. 507-510). IEEE.

[2]- Valera, A., Vallés, M., Díez, J. L., Pizá, R., & Sánchez, A. Utilización de Sensorización Externa en Robótica. XXIV Jornadas de Automática, España, ISBN, 84-931846.

[3] Sánchez, A., & Ramos, C. SEGUIMIENTO VISUAL DE OBJETOS UTILIZANDO TÉCNICAS DE PREDICCIÓN, XXI Jornadas de Automatica, (2000).

[4] Documentación de OpenCV [Internet] [actualizado Mar 2017, citado Jul 2017]
Disponible en: http://docs.opencv.org/3.2.0/

[5] Manual de referencia de OpenCV 3.0 [Internet] [actualizado Jun2014, citado Jun 2017]
Disponible en: http://docs.opencv.org/3.0-beta/opencv2refman.pdf

[6]- Sánchez, A., & Marchant, J. (1997). Fast and robust method for tracking crop rows using a two point Hough transform. Acts of BioRobotics, 97

[7]- Berti, E. M., Salmerón, A. J. S., & Benimeli, F. (2012). Kalman filter for tracking robotic arms using low cost 3D vision systems. In The Fifth International Conference on Advances in Computer-Human Interactions (pp. 236-240)

[8] Viala, C. R., Salmerón, A. J. S., & Fernández, R. S. (2003). Técnicas de calibrado de cámaras. Journal de Mathematiques Pures et Appliques, Leon.

[9] Ricolfe, C., & Sánchez, A. J. (2008). PROCEDIMIENTO COMPLETO PARA EL CALIBRADO DE CÁMARAS UTILIZANDO UNA PLANTILLA PLANA. RIAII, 5(1), 93-101.

[10] Viala, C. R., & Salmeron, A. S. (2004, June). Performance evaluation of linear camera calibration techniques. In Automation Congress, 2004. Proceedings. World (Vol. 18, pp. 49-54). IEEE.

[11] Ricolfe-Viala, C., & Sanchez-Salmeron, A. J. (2007). Improved camera calibration method based on a two-dimensional template. Pattern Recognition and Image Analysis, 420-427.

[12] Ricolfe-Viala, C., & Sanchez-Salmeron, A. J. (2011). Camera calibration under optimal conditions. Optics express, 19(11), 10769-10775.

[13] Ricolfe-Viala, C., & Sanchez-Salmeron, A. J. (2011, September). Optimal conditions for camera calibration using a planar template. In Image Processing (ICIP), 2011 18th IEEE International Conference on (pp. 853-856). IEEE.

ANEXOS

1. Manuales de uso

Al presente documento se adjuntan una serie de programas informáticos para *Windows*, elaborados para el ensayo de la eficacia de los sistemas de detección de objetos. Todos ellos siguen una estructura similar, debido a que se trata en realidad de un mismo algoritmo con diversas modificaciones para mejorar su rendimiento.

1.1. Conexionado

Cada una de las cámaras se conectará a uno de los puertos de la tarjeta de adquisición, en caso de disponer de más de una, se conectarán ambas a la misma tarjeta. La cámara Izquierda se conectará al Puerto A, y la Derecha al puerto B.

1.2. Ficheros adicionales

Para el correcto funcionamiento de los programas, son necesarios algunos ficheros que deberán estar ubicados en la misma carpeta que el ejecutable. Los ficheros *opencv_world320.dll* y *opencv_world320d.dll* contienen las librerías dinámicas de OpenCV, asimismo se debe incluir in fichero con nombre *StereoCalibration.yml* conteniendo los parámetros de calibración como se especifica en el siguiente apartado.

1.2.1. Fichero de calibración

Para agilizar la configuración del programa, los parámetros de calibración se leen desde un fichero externo, puesto que se trata de parámetros invariantes, que no dependen del programa sino de las características físicas de las cámaras.

El fichero se puede generar mediante un programa externo que ejecute la función de calibración estereoscópica de OpenCV *stereocalibrate()*. Se guardará en formato *YAML* (estándar de OpenCV, y con funciones específicas para el manejo de ficheros). En el fichero se almacenarán los valores devueltos por la función de calibración bajo los siguientes nombres de variables:

- *camMat1*: matriz intrínseca de la cámara izquierda.
- *camMat2*: ídem para la cámara derecha.
- *disCoe1*: coeficientes de distorsión de la cámara izquierda.
- *disCoe2*: ídem para la cámara derecha.
- *R*: matriz de rotación entre las cámaras.
- *T*: vector de traslación entre las cámaras.
- *E*: matriz esencial del sistema.
- *F*: matriz fundamental del sistema.

Para la calibración se recomienda utilizar dos o más pares de imágenes que contengan un patrón de calibración de tablero de ajedrez, con distinto número de filas y columnas, cada par mostrándolo en distinta posición y orientación.

1.2.2. Drivers adicionales

Asimismo también son necesarios los drivers correspondientes para el funcionamiento de la tarjeta de adquisición, proporcionados por el fabricante. Las cámaras empleadas no necesitan drivers específicos instalados en el sistema, pero sí es conveniente el uso del programa de configuración *Mikrotron Control Tool* para la introducción de los parámetros específicos. Alternativamente, se pueden configurar las cámaras mediante comunicación serie a través del conexionado con la tarjeta (*para más detalles acerca de los comandos de configuración de las cámaras véase el manual del fabricante*).

1.3. Configuración de las cámaras

Para el correcto funcionamiento de los programas es necesario configurar las cámaras en correspondencia a los parámetros que esperan los programas. A continuación se muestran los parámetros que se han configurado con sus respectivos valores introducidos. En ese manual se muestran las ventanas del programa *Mikrotron Control Tool* con los parametros que se han ajustado en cada una:

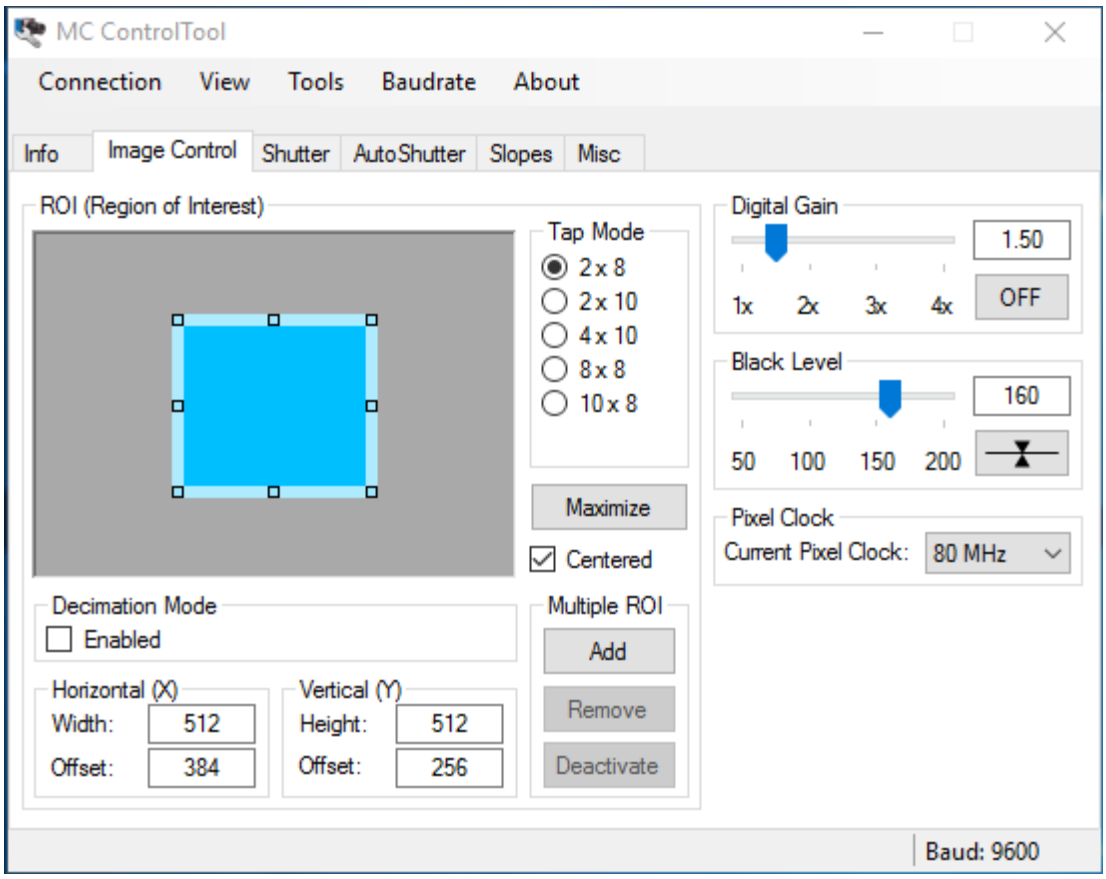


Fig. 1. MC Control Tool, pestaña Image Control.

En la pestaña de control de imagen se introduce:

- Tap mode: 2 x 8bits.
- Anchura: 512 píxeles.
- Altura: 512 píxeles.
- Centrado: Activado.
- Ganancia digital: 1.50 (opcional, el valor más adecuado depende del entorno)
- Balance de negros: 160 (+32).

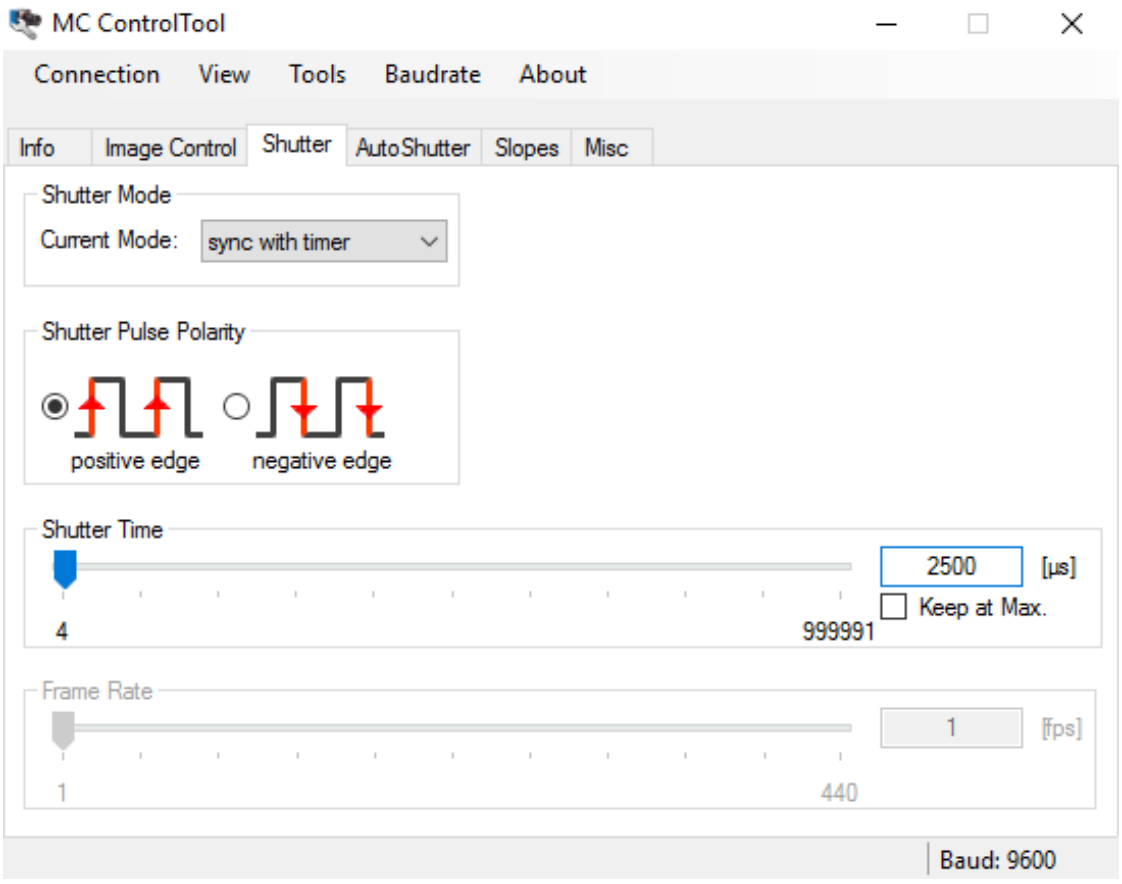


Fig. 2. MC Control Tool, pestaña Shutter.

En la pestaña de disparador:

- Modo: sincronismo con temporizador (o ancho de pulso si se configura la exposición mediante la tarjeta).
- Polaridad: flanco positivo.
- Tiempo de disparador - exposición: entre 2000 y 8000µs (dependiendo de la cantidad luz, mayor exposición si la luz es baja).

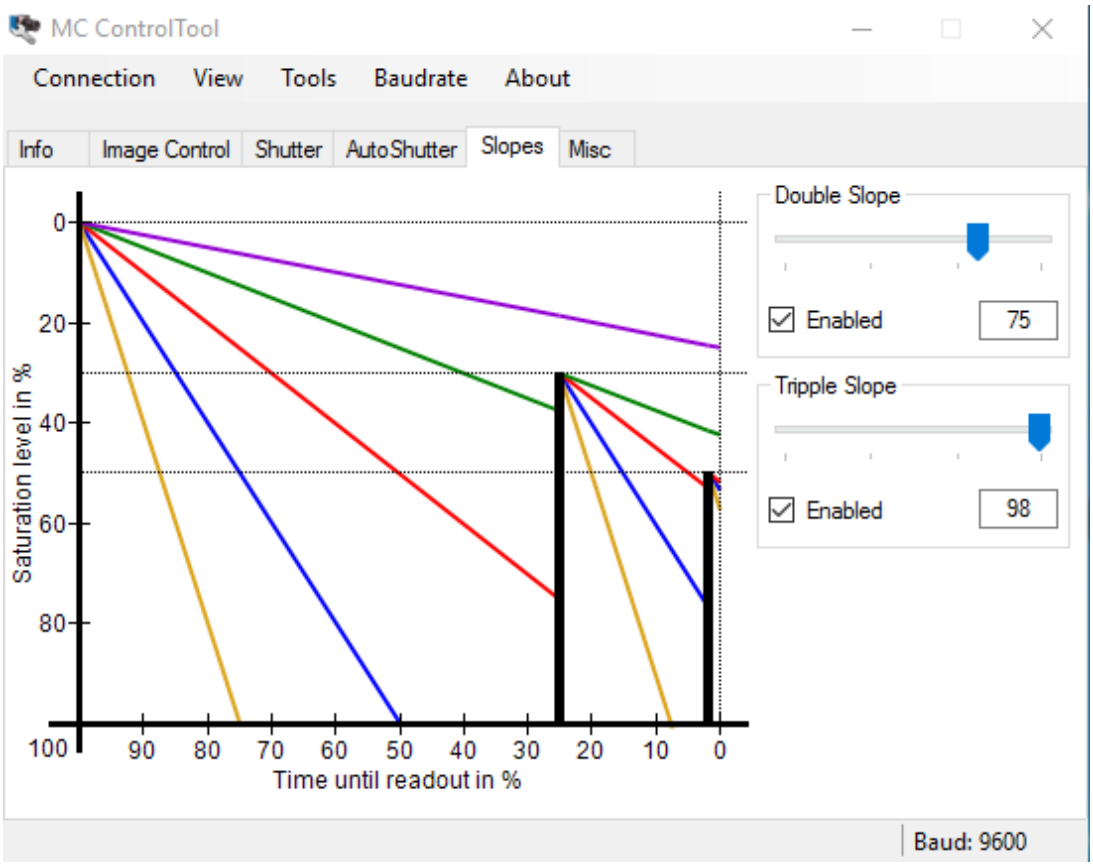


Fig. 3. MC Control Tool, pestaña Slopes.

En la pestaña de pendientes (opcional y recomendable):

- Doble pendiente: activa en 75%.
- Triple pendiente: activa en 98%.

1.4. Programas

Se incluyen un total de cinco programas correspondientes a los cinco algoritmos analizados.
Al inicio de cada programa se pide al usuario escoger el número de tarjeta:

"Choose board number [0 - x]"

Donde *x* es el número de tarjeta disponible correspondiente a la última tarjeta, empezando en 0. Si se dispone de una única tarjeta el único valor posible es 0. Si se dispone de más de una tarjeta reconocida, se deberá introducir número el correspondiente a la tarjeta en la que se hayan conectado las cámaras. Se valida el número pulsando *"Intro"*.

1.4.1. Tracking 1: método off-line

El programa número 1 muestra el método de procesado off-line, capturando un total de 100 pares de fotogramas (1 segundo de vídeo - 150Mb en RAM), procesando fotograma a fotograma y mostrando los resultados con un intervalo de10ms.

Una vez elegido el número de tarjeta, el programa realiza la configuración y muestra el mensaje:

"Pulse una tecla para continuar..."

Aparecerán dos ventanas que permiten la visualización de las imágenes capturadas durante la grabación. Teniendo activa la ventana del programa, pulsando cualquier tecla se inicia la captura del vídeo. Esta pausa se realiza con la intención de preparar al usuario para la grabación debido a la corta duración de la misma.

Cuando se completa la grabación aparecen dos nuevas ventanas mostrando los resultados. En ellas aparecerá un círculo rojo señalando donde se ha detectado la esfera. En caso de detectarla en ambas imágenes, aparecen por la consola las coordenadas espaciales donde se ha triangulado el objeto.

Al alcanzar el último fotograma, el programa termina, espera a que el usuario revise los resultados y se cierra al pulsar una tecla.

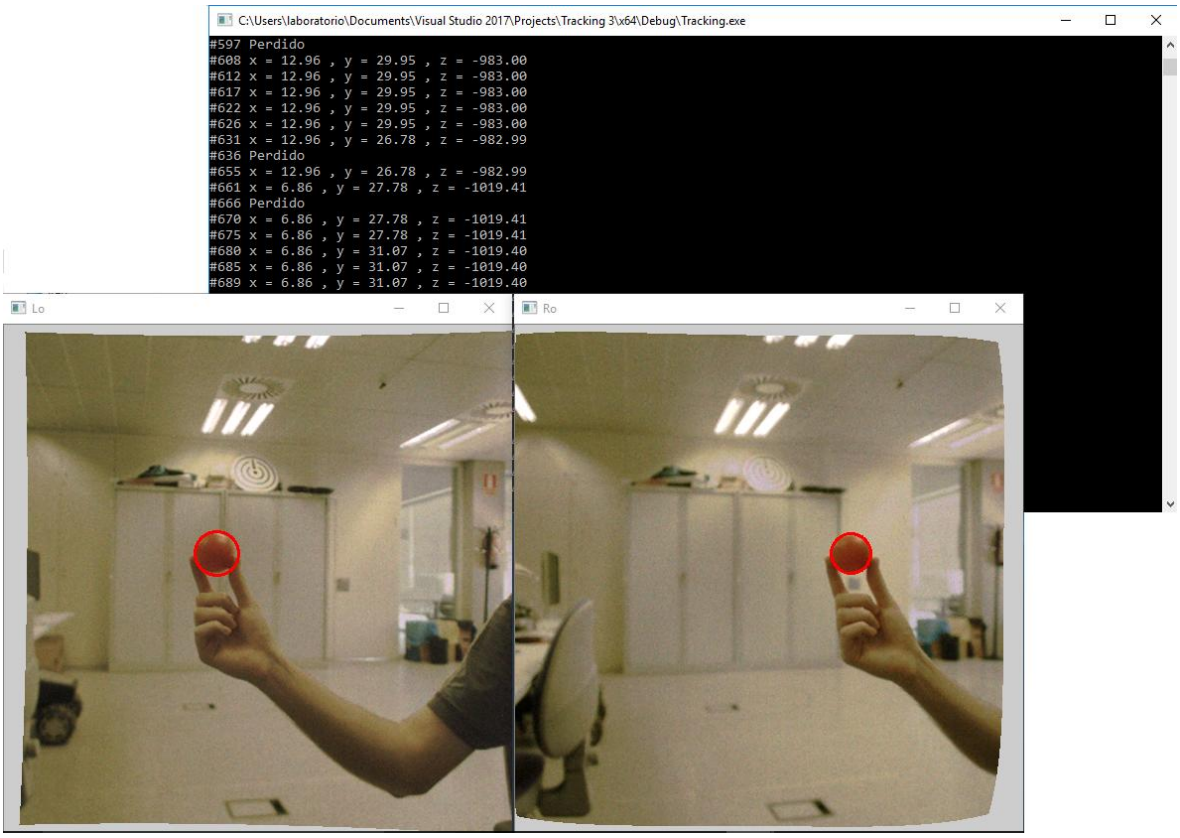


Fig. 1. Funcionamiento de los programas 1 y 2.

1.4.2. Tracking 2: método on-line

El programa número 2 es una adaptación del programa número 1 para funcionar en tiempo real, y al contrario que en el caso anterior, funciona indefinidamente hasta que el usuario decida terminar.
Al igual que en el caso anterior, después de introducir el número de tarjeta aparecen dos ventanas para la visualización de los resultados. En este caso, el programa también espera a que el usuario pulse una tecla para continuar, pero esta vez manteniendo activa una de las ventanas de visualización.
Esto se debe a que en el programa 1 las ventanas son de visualización de la tarjeta de captura, que no admiten comandos. En el programa 2 las ventanas son de OpenCV que sí los admiten.
Los resultados se muestran de forma similar al caso anterior: un círculo indicando la forma detectada y en la consola, las coordenadas en forma de texto, acompañadas del número de fotograma del que se han obtenido. Si no se detecta la esfera en ambas imágenes, no se puede devolver un resultado triangulado y se devuelve por consola el mensaje *"Perdido"* acompañado por el número del primer fotograma con resultado negativo. La consola no devuelve más resultados hasta que no se vuelve a detectar el objeto.
Para terminar el programa, se pulsa la tecla *"Esc"* manteniendo una de las ventanas de visualización activas. El programa muestra el mensaje por consola *"Pulse una tecla para continuar..."* esperando cualquier tecla para cerrar el programa. Esto permite revisar los resultados sin introducir valores nuevos.

1.4.3. Tracking 3: modelo de predicción

El programa número 3 incluye un modelo de predicción de la posición del objeto a seguir basado en la dinámica del sólido libre, y busca en un cuadrado de 120x120 píxeles alrededor del punto obtenido de la predicción.
El manejo del programa es idéntico al caso anterior: se introduce el número de tarjeta, cuando la configuración del programa termina se espera una tecla para empezar y la tecla *"Esc"* para terminar.
Los resultados se muestran igualmente en las ventanas de visualización y por consola. El círculo representa la forma reconocida y el cuadrado verde indica la zona en la que se busca el objeto, en el centro de este hay un punto naranja que indica el punto de la predicción. La ubicación espacial se muestra por consola junto con el número de fotograma.
Las predicciones aparecen después de dos predicciones consecutivas, es normal que en situaciones en las que la detección sea más difícil (por ocultaciones, baja iluminación, etc.) el cuadrado desaparezca habitualmente. Esto indica que el modelo está resultando poco efectivo y tiene frecuentes caídas en la velocidad por procesar frecuentemente la imagen completa.

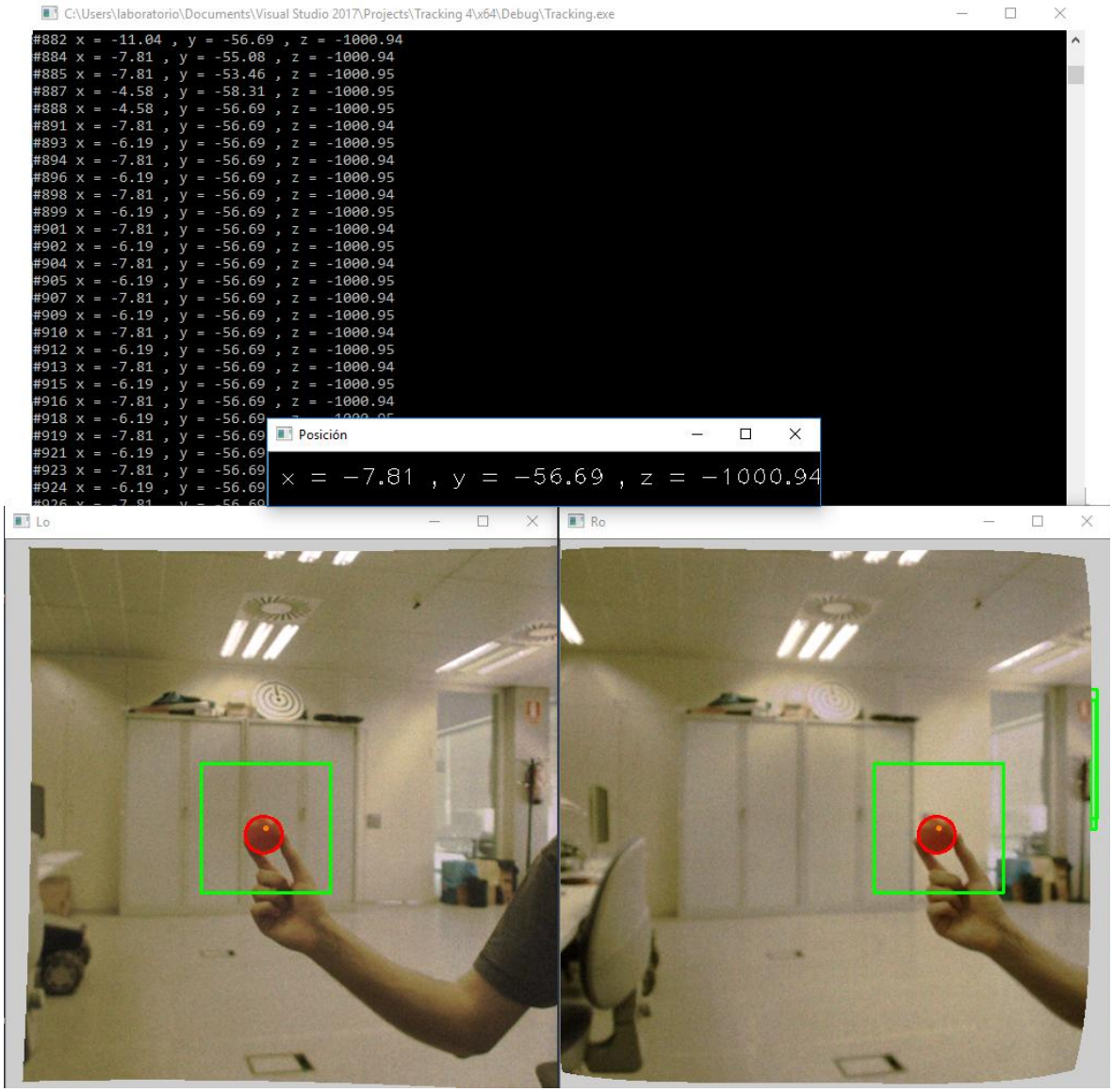


Fig. 2. Funcionamiento del programa 3.

1.4.4. Tracking 4: predicción estática

El programa número 4 incluye un método para compensar las situaciones en las que se puede perder momentáneamente el objeto a detectar, suponiendo el objeto en la última posición conocida y ampliando el área de búsqueda.

El manejo del programa es igual a los casos anteriores. Este programa muestra una tercera ventana de visualización, por la que se muestra la posición en el espacio obtenida. La diferencia con el valor mostrado por consola consiste en que si no se detecta el objeto a seguir, la consola muestra el mensaje *"Perdido"*, mientras que la ventana muestra la posición supuesta acompañada de un *"?"*.

La representación visual es igual que en los casos anteriores: el círculo rojo representa la forma detectada, el punto naranja representa el punto de la predicción, y el cuadrado representa la zona de la imagen en la que se está buscando el objeto, que será de color verde si se basa en los dos últimos resultados positivos o amarillo si se basa en suposiciones.
Si el cuadrado es de color verde tiene un tamaño de 120x120 píxeles y está basado en medidas reales obtenidas, en cambio, si es de color amarillo, va aumentando de dimensiones según se obtienen resultados negativos.

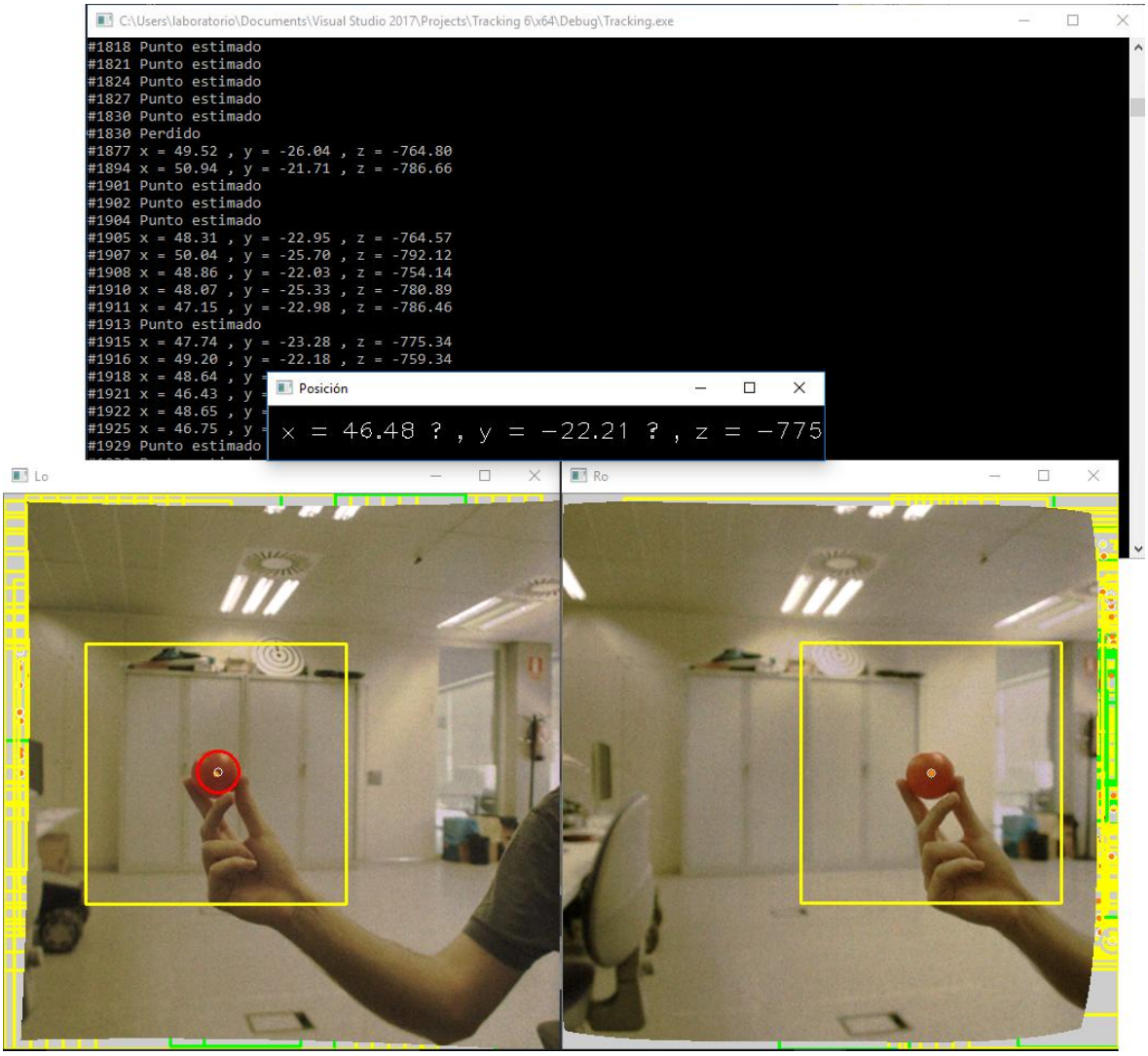


Fig. 3. Funcionamiento del programa 4.

El funcionamiento normal de este programa incluye instantes (1 - 4 fotogramas) en los que se pierde el objeto, el aumento del área de búsqueda es claramente apreciable por lo que es fácil apreciar en cuantos fotogramas seguidos se ha basado el área de búsqueda en suposiciones. Si se aprecia que el cuadrado de búsqueda tiende a hacerse grande habitualmente o incluso a alcanzar su tamaño máximo y desaparecer, indica que el modelo no funciona correctamente y las detecciones no son suficientemente próximas en el tiempo como para considerar la aproximación del modelo como fiable.

1.4.5. Tracking 5: filtro de Kalman

El programa número 5 está basado en el programa 4, pero en lugar de suponer el objeto estático cuando se pierde, se estima una trayectoria mediante un filtro de Kalman. El filtro se actualiza con cada valor obtenido y calcula una próxima posición mediante predicción, si se pierde la esfera, la predicción se convierte en el nuevo valor introducido en el filtro y se usa para estimar la siguiente posición.

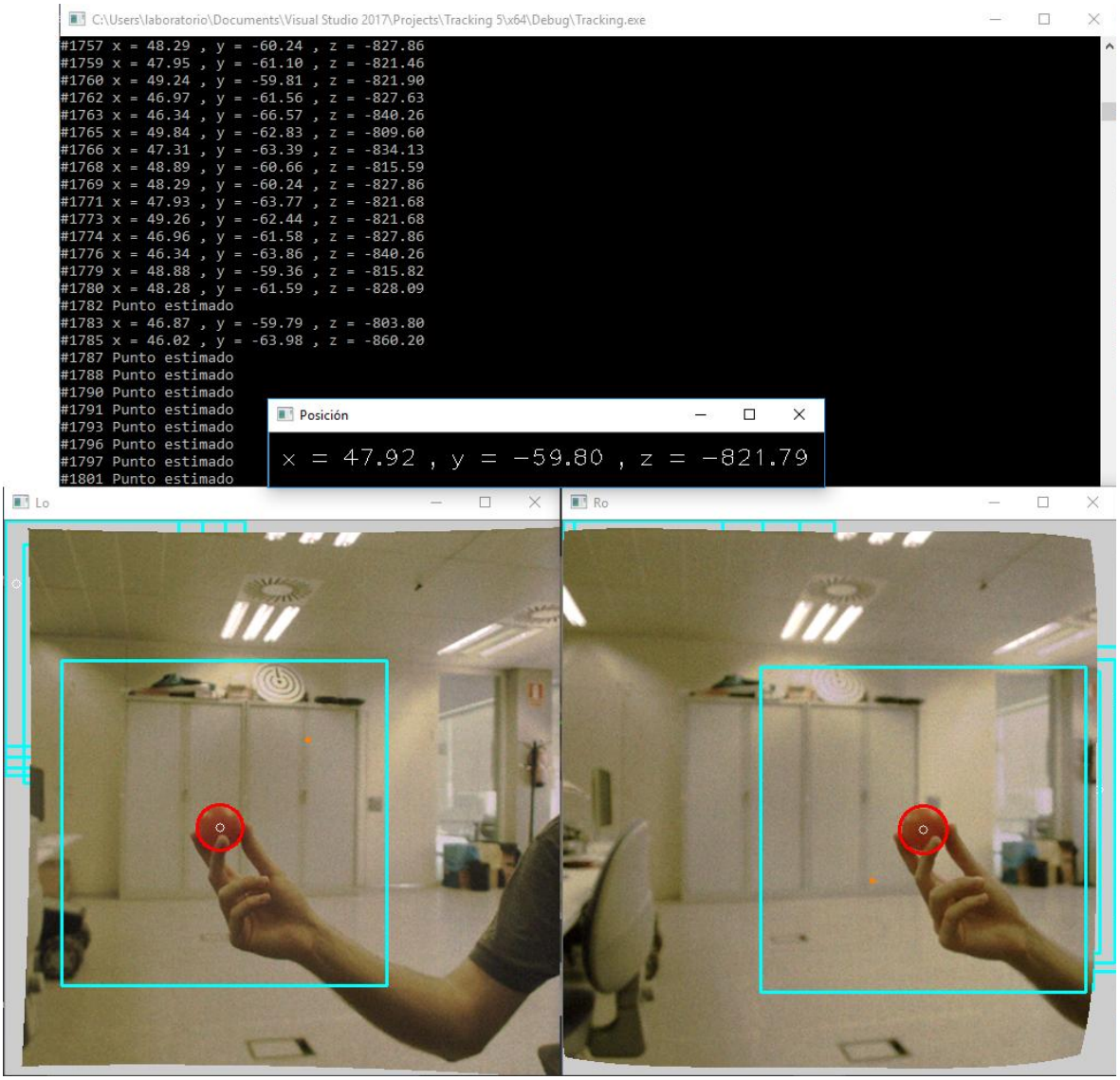


Fig. 4. Funcionamiento del programa 5.

Una vez más el manejo del programa es idéntico a los anteriores.

La consola muestra los valores obtenidos por medidas reales, y *"Perdido"* si no se ha detectado el objeto y el número de fotograma. Una ventana de visualización muestra la ubicación en el espacio del objeto a seguir, acompañado de *"?"* si se ha obtenido por predicción.

Las otras dos ventanas muestran las imágenes procesadas. La forma detectada aparece como un círculo rojo, el punto obtenido del modelo como un punto naranja y el punto actual del filtro de Kalman como un círculo blanco.

La zona en la que se busca el objeto aparece marcada con un cuadrado verde si se obtiene del modelo, o cian si se centra en la predicción del filtro de Kalman.
Este algoritmo funciona correctamente cuando el objeto a seguir mantiene aceleraciones aproximadamente constantes, independientemente de si se oculta o no durante espacios de tiempo relativamente altos (5 - 9 fotogramas). El algoritmo tiende a fallar en casos en los que el objeto a seguir sufre cambios fuertes en la aceleración, esto es visible cuando al sufrir el objeto una aceleración, el área de búsqueda continúa moviéndose a lo largo de la trayectoria que mantenía, en lugar de seguir la nueva.

2. Manual de programación

Se anexa también el código fuente de los distintos programas, a continuación se presenta una descripción de los mismos. Se ha optado por describir los elementos que componen cada programa en lugar de descripciones de los programas completos debido a que son muy similares y comparten gran cantidad de código. La mayoría de las funciones se realizan para las dos imágenes del par, aunque el ejemplo muestra solo una de ellas, la segunda es análoga.

Los programas emplean las siguientes librerías:

```
//Librerías de I-O
    #include <stdio.h>
    #include <iostream>

//Librerías de OpenCV
    #include <opencv2/core.hpp>          //Funciones básicas
    #include <opencv2/imgcodecs.hpp>     //Operaciones con archivos de
    imagen
    #include <opencv2/highgui.hpp>       //Ventanas de salida de imagen
    #include <opencv2 /calib3d.hpp>      //Reconstrucción 3d
    #include <opencv2/tracking.hpp>      //Seguimiento de objetos

//Librerías de Silicon Software
    #include <sisoIo.h>                  //Funciones I-O
    #include" board_and_dll_chooser.h"   //Función selector de tarjeta
    #include <fgrab_struct.h>            //Funciones de la capturadora
    #include <fgrab_prototyp.h>
    #include <fgrab_define.h>
    #include <SisoDisplay.h>             //Ventanas de previsualización
```

Todos los códigos comienzan con una definición de gran parte de las variables que se van a utilizar.

A continuación se leen los parámetros de calibración de las cámaras desde el fichero, y se realiza la corrección de imagen obteniendo los parámetros necesarios:

```
//Contenidos del fichero de calibración
    cv::FileStorage fs("StereoCalibration.yml", cv::FileStorage::READ);
    fs["camMat1"] >> camMat1;             //Matriz cámara izq.
    fs["camMat2"] >> camMat2;             //Matriz cámara dch.
    fs["disCoe1"] >> disCoe1;             //Coef distorsión cam. izq.
    fs["disCoe2"] >> disCoe2;             //Coef distorsión cam. dch.
    fs["R"] >> R;                         //Vector de rotación
    fs["T"] >> T;                         //Vector de traslación
    fs["E"] >> E;                         //Matriz esencial
    fs["F"] >> F;                         //Matriz fundamental
    fs.release();
```

```
//Corrección de la cámara
    cv::stereoRectify(camMat1, disCoe1, camMat2, disCoe2, size, R, T, R1,
R2, P1, P2, Q, CV_CALIB_ZERO_DISPARITY, -1, cv::Size(0, 0));
    cv::initUndistortRectifyMap(camMat1, disCoe1, R1, P1, size, CV_32FC1,
outL1, outL2);
    cv::initUndistortRectifyMap(camMat2, disCoe2, R2, P2, size, CV_32FC1,
outR1, outR2);
```

El siguiente elemento común del código es la inicialización de la tarjeta, es un fragmento relativamente largo en el que se introducen todos los parámetros necesarios. Se comprueba que todos los parámetros se han introducido correctamente con la un *if* evaluando la función para introducir cada parámetro. Si la función se ejecuta correctamente el programa continúa, pero si devuelve un error, se muestra el error y se termina el programa.
Este proceso se inicia con la detección de la tarjeta y la inicialización de la *applet* correspondiente. El código dispone de casos para todas las tarjetas soportadas. En caso de utilizar un modelo distinto, se puede introducir el nombre de la *applet* correspondiente al modelo.

```
const char *applet;
    switch (Fg_getBoardType(boardNr)) {
    [...]
    case PN_MICROENABLE4AD4CL:
     applet = "Acq_DualBaseAreaBayer8";
          break;
    [...]
    }

    if ((fg = Fg_Init(applet, boardNr)) == NULL) {
          fprintf(stderr,"error in Fg_Init:%s\n", Fg_getLastErrorDescription(NULL));
          system("pause");
          return FG_ERROR;
    }
```

En caso de utilizar las ventanas de previsualización de las librerías de la tarjeta *SisoDisplay*, para asignar un espacio de memoria se incluye:

```
    int dispId0 = CreateDisplay(8 * bytePerSample * samplePerPixel,
width, height);
    SetBufferWidth(dispId0, width, height);
    int dispId1 = CreateDisplay(8 * bytePerSample * samplePerPixel,
width, height);
    SetBufferWidth(dispId1, width, height);
```

Seguidamente, se asigna un espacio de memoria para los dos buffers en anillo donde se almacenarán las imágenes. Todos los parámetros (y los buffers de memoria) están duplicados, puesto que las imágenes se adquieren por parejas, izquierda y derecha.

```
size_t totalBufferSize = width * height * samplePerPixel * bytePerSample
* nbBuffers;
    dma_mem *memHandle1 = Fg_AllocMemEx(fg, totalBufferSize, nbBuffers);
    if (memHandle1 == NULL) {
          fprintf(stderr, "error in Fg_AllocMemEx: %s\n",
          Fg_getLastErrorDescription(fg));
          CloseDisplay(dispId0);
          CloseDisplay(dispId1);
          Fg_FreeGrabber(fg);
          system("pause");
          return FG_ERROR;
    }
    dma_mem *memHandle2 = Fg_AllocMemEx(fg, totalBufferSize, nbBuffers);
    if (memHandle2 == NULL) {
          fprintf(stderr, "error in Fg_AllocMemEx: %s\n",
          Fg_getLastErrorDescription(fg));
          CloseDisplay(dispId0);
          CloseDisplay(dispId1);
          Fg_FreeGrabber(fg);
          system("pause");
          return FG_ERROR;
    }
```

A continuación se muestra un ejemplo de introducción de uno de los parámetros. La lista de parámetros necesarios es:

- Anchura de imagen: 512 píxeles
- Altura de imagen: 512 píxeles
- Formato de envío de la imagen: Dual Tap 8 bit
- Usar DVal: Sí
- Tipo de filtro Bayer: Verde seguido de azul (GreenFollowedByBlue)
- Trigger mode: Generator (utilizar señal de disparo interno)
- Trigger state: Active
- Velocidad de disparador: 100fps
- Ancho de pulso (opcional si se configura el disparo por ancho de pulso en las cámaras, el valor que se introduciría será el que se haya ajustado de tiempo de disparo en las cámaras, impide el uso de pendientes múltiples.)
- CC Mapping: Select0 = CC PulseGen0 (señal de disparo a través del canal 0)
- Offset de ganancia: 1.1 para canales rojo y azul (corrección de color)
- Bit Alignment: left

Los parámetros deben introducirse por duplicado, uno por cada puerto, puesto que se pueden utilizar configuraciones distintas en cada cámara. Esto es posible para aplicaciones en las que cada cámara se utilice para un proceso distinto con características diferentes. En este caso, ambas cámaras funcionan para la misma aplicación, por lo que es muy conveniente que funcionen bajo la misma configuración.

```
MeCameraLinkFormat camtype = FG_CL_DUALTAP_8_BIT;
    if (Fg_setParameter(fg,FG_CAMERA_LINK_CAMTYPE,&camtype,camPortL) < 0)
{
          fprintf(stderr, "Fg_setParameter(FG_CAMERA_LINK_CAMTYPE)
          failed: %s\n", Fg_getLastErrorDescription(fg));
          Fg_FreeMemEx(fg, memHandle1);
          Fg_FreeMemEx(fg, memHandle2);
          CloseDisplay(dispId0);
          CloseDisplay(dispId1);
          Fg_FreeGrabber(fg);
          system("pause");
          return FG_ERROR;
    }
```

Una vez configurado el sistema, se puede comenzar a capturar imágenes. En primer lugar se preparan las ventanas de visualización necesarias:

```
    cv::namedWindow("Lo", cv::WINDOW_AUTOSIZE); //Output izquierda
    cv::namedWindow("Ro", cv::WINDOW_AUTOSIZE); //Output derecha
    cv::imshow("Posición", bgr);                //Posición texto
```

Para iniciar el proceso se evalúa la función *Fg_AcquireEx()* igual que las funciones de introducción de parámetro para terminar el programa en caso de error.

```
if   ((Fg_AcquireEx(fg,   camPortL,   nrOfPicturesToGrab,   ACQ_STANDARD,
memHandle1))   ||   (Fg_AcquireEx(fg,   camPortR,   nrOfPicturesToGrab,
ACQ_STANDARD, memHandle2)) < 0)
          {
                fprintf(stderr, "Fg_AcquireEx() failed: %s\n",
                Fg_getLastErrorDescription(fg));
                Fg_FreeMemEx(fg, memHandle1);
                Fg_FreeMemEx(fg, memHandle2);
                CloseDisplay(dispId0);
                Fg_FreeGrabber(fg);
                system("pause");
                return FG_ERROR;
          }
```

Una vez el sistema comienza a capturar, el programa entra en un bucle *while* esperando a terminar el proceso. En el algoritmo de procesado off-line se espera a alcanzar el número máximo de fotogramas, en los algoritmos on-line, la función de OpenCV *WaitKey(t)* refresca las imágenes mostradas cada *t* milisegundos y espera una tecla. El programa sale del bucle cuando la función detecta que se ha pulsado la tecla *Esc*.

La función *Fg_getLastPicNumberBlockingEx()* devuelve el número del último fotograma capturado.

A continuación se copian los datos desde un puntero que señala al fotograma actual al formato de OpenCV para trabajar con ellos:

```
imageL = cv::Mat(height, width, CV_8UC3, Fg_getImagePtrEx(fg, cur_pic_nr, camPortL, memHandle2)).clone();
```

A partir de este punto comienza el procesado de OpenCV. En el método off-line tiene lugar en un segundo bucle, mientras que en los métodos on-line tiene lugar en el mismo bucle que la captura.

En primer lugar se compensa la distorsión de la imagen utilizando los parámetros obtenidos mediante la calibración:

```
cv::remap
(imageL, outL, outL1, outL2, cv::INTER_LINEAR,cv::BORDER_TRANSPARENT,0);
cv::remap
(imageR, outR, outR1, outR2, cv::INTER_LINEAR,cv::BORDER_TRANSPARENT,0);
```

En los algoritmos más básicos en los que se trabaja con la imagen completa se opera bajo un único caso, que es procesar toda la imagen. En los algoritmos en los que se utiliza una predicción se establecen los diferentes casos en los que se disponga o no de una predicción.

Los casos son:

- Se dispone de dos valores anteriores: se usan estos valores para calcular la ubicación probable del objeto en el fotograma actual.
- No se dispone de dos valores anteriores o se han analizado una predicción (algoritmo 3) o 10 (algoritmos 4 y 5) sin éxito: se analiza la imagen completa. Equivalente al caso único de los algoritmos básicos.
- En los programas 4 y 5 existe un tercer caso en el que no se ha detectado el objeto, pero se toma el último valor conocido (algoritmo 4) o estimado por el filtro de Kalman (algoritmo 5) hasta 10 fotogramas seguidos. En estos casos se analiza la zona de la predicción aumentando el área cada vez.

Si se analiza la imagen completa, se separa la imagen por canales y se realiza el filtrado de color:

```
cv::split(outL, canales);
goutL = 4 * (canales[2] - 0.55*(canales[0] + canales[1])); //img B/N
cv::blur(goutL, goutL, cv::Size(3, 3));  //Suavizar bordes
```

Si se dispone de valores anteriores se predice el punto siguiendo el modelo y se reproyecta sobre la imagen:

```
PointEst[0].y = x + ((x - PointPre[0].x) / (timeCur))*timeEst;
PointEst[0].x = y + ((y - PointPre[0].y) / (timeCur))*timeEst + 0.004950
* (timeEst)*(timeEst);
PointEst[0].z = z + ((z - PointPre[0].z) / (timeCur))*timeEst;

cv::projectPoints(PointEst, r1, to1, K1, disCoe1, mcLEst);
cv::projectPoints(PointEst, r2, to2, K2, disCoe2, mcREst);
```

Se han invertido los ejes x e y debido a la forma en la que OpenCV trata las imágenes en color intercambiando los ejes. Al cambiar los valores, el punto aparece en la posición correcta en la reproyección.

En los casos en los que se emplea la predicción, hay unas líneas de código después de la corrección de la cámara:

```
to1[0].x = T1.at<double>(0,0) / T1.at<double>(3, 0);
to1[0].y = T1.at<double>(1, 0) / T1.at<double>(3, 0);
to1[0].z = T1.at<double>(2, 0) / T1.at<double>(3, 0);
[...]
cv::Rodrigues(Ro1, r1);
```

Estas funciones se utilizan para convertir los parámetros que se utilizan para triangular el objeto en el espacio en parámetros que permitan realizar la función inversa.

Se determinan las esquinas superior izquierda e inferior derecha del área de búsqueda a 60 píxeles de distancia del punto predicho, limitando su valor si excede los límites de la imagen. Se realiza el mismo filtrado que en el caso de procesar la imagen completa, pero tomando solo un rectángulo de la imagen completa:

```
ROIR = outR(cv::Rect(topleftR, botright));
```

Los tiempos se obtienen con las funciones *StartCounter()* y *GetCounter()* que restan los valores del reloj del sistema entre dos instantes determinados y devuelven el intervalo en microsegundos. El

temporizador se detiene cuando se realiza el cálculo de la predicción y se pone a contar inmediatamente, devolviendo así el espacio de tiempo entre una predicción y la siguiente.
Tanto si se analiza la imagen completa como solo una región, se utiliza la función de detección de círculos *HoughCircles()* en la zona a buscar. Se introducen los parámetros siguientes:

- Ratio inverso: 4
- Distancia mínima entre centros: 200px
- Umbral de detección de bordes: 200
- Umbral de detección de centros: 20
- Sin radio mínimo
- Radio máximo 40px

```
cv::HoughCircles(ROIL, circlesL, CV_HOUGH_GRADIENT, 4,200,200,20,0, 40);
```

A continuación se busca un círculo detectado con color rojo en su centro. Con un bucle *for* se recorren todos los resultados obtenidos por la función *HoughCircles()*.

```
for (size_t i = 0; i < circlesL.size(); i++)
	{
		cv::Point center(cvRound(circlesL[i][0]),
		cvRound(circlesL[i][1]));
		int radius = cvRound(circlesL[i][2]);
		cv::Scalar colour;
		if (found < 2 && predictions >= maxPred)// toda la imagen
		{
			colour = goutL.at<uchar>(cvRound(circlesL[i][1]),
			cvRound(circlesL[i][0]));
		}
		else //Solo una zona
		{
			colour = ROIL.at<uchar>(cvRound(circlesL[i][1]),
			cvRound(circlesL[i][0]));

		}
//Si el círculo detectado tiene rojo en su centro y está dentro de los límites de la imagen
		if ((colour.val[0] > 250) && ((cvRound(circlesL[i][0]) + 2 * radius < 512) && ((cvRound(circlesL[i][1]) + 2 * radius) < 512)))
		{
			mcL[0] = cv::Point2f(circlesL[i][0],
			circlesL[i][1]); //Almacenar este valor
			if (found >= 2 || predictions < maxPred) {
			//Si analizamos ROI, añadir su offset
				mcL[0].x = mcL[0].x + topleftL.x;
```

```
                        mcL[0].y = mcL[0].y + topleftL.y;
                }
                cv::circle(outL, mcL[0], radius, cv::Scalar(0, 0,
                255), 2, 1);
                OKL = true; //resultado positivo
                break;
            }
            else
                OKL = false; // resultado negativo, ninguno cumple
        }
```

Nótese que se han invertido los ejes a la hora de pasar desde los puntos obtenidos por la detección de círculos a la variable donde se almacena el resultado.

Se repite el proceso para la otra imagen del par, y en caso de obtener resultado positivo en ambas, se utilizan los resultados para triangular el punto en el espacio.

```
if (OKL == true && OKR == true)
        {
//Antes de actualizar el valor, almacenar el valor anterior para
cálculos
            PointPre[0].x = x;
            PointPre[0].y = y;
            PointPre[0].z = z;
            lost = false;
//Correción de los valores obtenidos
            cv::correctMatches(F, mcL, mcR, mcL, mcR);
//Función de triangulación
            cv::triangulatePoints(P1, P2, mcL, mcR, PointCur);
//Extracción de los valores
            w = PointCur.at<float>(3, 0);
            x = PointCur.at<float>(0, 0) / w;
            y = PointCur.at<float>(1, 0) / w;
            z = PointCur.at<float>(2, 0) / w;

//Muestra los valores obtenidos
            bgr = cv::Mat::zeros(50, 512, CV_8U);
            char str[75];
            sprintf(str, "x = %.2f , y = %.2f , z = %.2f", x, y, z);
```

```
                cv::putText(bgr, str, cv::Point2f(10, 30),
                cv::FONT_HERSHEY_PLAIN, 1.5, cv::Scalar(255));
                printf("#%Ii x = %.2f , y = %.2f , z = %.2f\n",
                cur_pic_nr, x, y, z);
                found++;
          }
//Resultado anterior positivo y actual negativo
else if (lost == false)
          {
                printf("#%Ii Perdido\n", cur_pic_nr);
                lost = true;
                found = 0;
          }
//Refresco de las ventanas de visualización
          cv::imshow("Lo", outL);
          cv::imshow("Ro", outR);
          cv::imshow("Posición", bgr);
//Almacena intervalo de tiempo anterior antes de calcular el siguiente
          timeCur = timeEst;
          }
```

Para el cálculo de tiempos mencionado anteriormente se han incluido dos funciones que inicializan y leen respectivamente un cronómetro basado en el reloj del sistema. De este modo se puede obtener con una precisión elevada el tiempo transcurrido entre dos marcadores temporales, esto es, entre dos fotogramas consecutivos procesados.

```
void StartCounter()
{
    LARGE_INTEGER li;
    if (!QueryPerformanceFrequency(&li))
          std::cout << "QueryPerformanceFrequency failed!\n";

    PCFreq = double(li.QuadPart) / 1000.0;

    QueryPerformanceCounter(&li);
    CounterStart = li.QuadPart;
}
double GetCounter()
{
    LARGE_INTEGER li;
    QueryPerformanceCounter(&li);
    return double(li.QuadPart - CounterStart) / PCFreq;
}
```

El algoritmo con filtro de Kalman incluye la inicialización del filtro antes de comenzar a capturar imágenes. El filtro se inicializa con una matriz que modeliza posición, velocidad y aceleración:

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0.5 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Se inicializa el filtro con posición en el centro de la imagen y velocidad y aceleración nulas:

```
KFL.statePre.at<float>(0) = 256;
KFL.statePre.at<float>(1) = 256;
KFL.statePre.at<float>(2) = 0;
KFL.statePre.at<float>(3) = 0;
KFL.statePre.at<float>(4) = 0;
KFL.statePre.at<float>(5) = 0;
```

Los otros parámetros a inicializar son la matriz de medidas que se inicializa a una matriz identidad, la matriz de covarianzas del ruido de proceso, que se inicializa a matriz identidad de valor $1 \cdot 10^{-4}$, la matriz de covarianza del ruido de las medidas, que se inicializa tambien a una matriz identidad unitaria, y la matriz de covarianza de los resultados que se inicializa a una matriz unitaria de valor 0.1.

Cuando se obtiene un resultado, se actualiza el filtro añadiendo el nuevo valor.

```
KFL.predict();
measurementL(0) = mcL[0].x;
measurementL(1) = mcL[0].y;
cv::Mat estimatedL = KFL.correct(measurementL);
```

Se añade un caso en los resultados para la situación en la que el resultado a mostrar sea la predicción del filtro, actualizando el valor con el resultado de la predicción.

```
predictionL = KFL.predict();
predictPtL.x = predictionL.at<float>(0);
predictPtL.y = predictionL.at<float>(1);

KFL.statePre.copyTo(KFL.statePost);
KFL.errorCovPre.copyTo(KFL.errorCovPost);
[...]
```

```
    std::vector<cv::Point2f> predictPtLArr(1);
    predictPtLArr[0] = predictPtL;
    [...]
    cv::correctMatches(F, predictPtLArr, predictPtRArr, predictPtLArr,
predictPtRArr);
    cv::triangulatePoints(P1,    P2,    predictPtLArr,    predictPtRArr,
PointCur);

    w = PointCur.at<float>(3, 0);
    x = PointCur.at<float>(0, 0) / w;
    y = PointCur.at<float>(1, 0) / w;
    z = PointCur.at<float>(2, 0) / w;
```

Cuando termina la captura se utiliza la función para detener la captura y se cierran todas las ventanas de visualización:

```
    Fg_stopAcquire(fg, camPortL);
    Fg_stopAcquire(fg, camPortR);
    cv::destroyAllWindows();     //Cierra las ventanas de OpenCV
    CloseDisplay(dispId0);       //Cierra las ventanas de la capturadora
    CloseDisplay(dispId1);
    printf("\n\nTerminado\n\n");
    system("pause");             //Para revisar resultados
```

A lo largo del código aparecen en varias ocasiones las funciones *cv::circle()* y *cv::rectangle()*. Estas funciones no desempeñan ninguna función real a la hora de calcular la posición del objeto, son funciones de control que representan visualmente los resultados obtenidos: el contorno y el centro del círculo detectado, el área de búsqueda o la proyección del punto obtenido mediante el modelo de predicción.

PRESUPUESTO

Cuadro de materiales

Núm	Denominación del material	Precio	T. amortizac. - T. uso	Cantidad	Total
1	Ordenador	1750	5 años - 3 meses	1	87.50
2	Cámara MC1363	3750	10 años - 3 meses	2	187.50
3	Tarjeta Micro Enable IV AD4-CL	850	10 años - 3 meses	1	21.25
4	Costes indirectos			10%	29.63
				Total materiales:	325.88

Cuadro de mano de obra

Núm	Denominación de la mano de obra	Precio	Cantidad	Total
1	Ingeniero Industrial	32.50	200	6500
			Total mano de obra:	6500

Anexo de justificación de precios

Núm	Cantidad	Denominación del material	Precio	Total
1	Diseño y evaluación de un sistema de seguimiento de objetos mediante cámaras de alta velocidad			
	200h	Ingeniero Industrial	32.50	6500
	3 meses	Ordenador	1750	87.50
	3 meses	Cámaras MC1363	3750	187.50
	3 meses	Tarjeta Micro Enable IV AD4-CL	850	21.25
	5.00%	Medios auxiliares		29.63
			Total:	6825.88

Son SEIS MIL OCHOCIENTOS VEINTICINCO EUROS CON OCHENTA Y OCHO CÉNTIMOS.

Presupuesto de ejecución por contrata

Proyecto:
Diseño y evaluación de un sistema de seguimiento de objetos mediante cámaras de alta velocidad

	Importe
Presupuesto de ejecución material	6825.88
13% de gastos generales	887.36
Suma	7713.24
21% de IVA	1619.78
Presupuesto de ejecución por contrata	9333.02

Asciende el presupuesto de ejecución por contrata a la expresada cantidad de NUEVE MIL TRESCIENTOS TREINTA Y TRES EUROS CON DOS CÉNTIMOS.