CAPÍTULO 1

LSD: “Line Segment Detection”

1.1. Introducción

LSD es un algoritmo de detección de segmentos publicado por Rafael Grompone von Gioi, Jérémie Jakubowicz, Jean-Michel Morel y Gregory Randall en abril de 2010. Es temporalmente lineal, tiene presición inferior a un píxel y no requiere de un tuneo previo de parámetros, como casi todos los demás algoritmos de idéntica función; puede ser considerado el estado del arte en cuanto a detección de segmentos en imágenes digitales. Como cualquier otro algoritmo de detección de segmentos, LSD basa su estudio en la búsqueda de contornos angostos dentro de la imagen. Estos son regiones en donde el nivel de brillo de la imagen cambia notoriamente entre píxeles vecinos, por lo que el gradiente de la misma resulta de vital importancia. Se genera previo al análisis de la imagen, un campo de orientaciones asociadas a cada uno de los píxeles denominado por los autores *level-line orientation field*. Dicho campo se obtiene de calcular las orientaciones ortogonales a los ángulos asociados al gradiente del la imagen. Luego, LSD puede verse como una composición de tres pasos:

(1) División de la imagen en las llamadas *line-support regions*, que son grupos conexos de píxeles con idéntica orientación, hasta cierta tolerancia.

(2) Búsqueda del segmento que mejor aproxime cada *line-support region*: aproximación de las regiones por rectángulos.

(3) Validación o no de cada segmento detectado en el punto anterior.

Los puntos (1) y (2) están basados en el algoritmo de detección de segmentos de Burns, Hanson y Riseman y el punto (3) es una adaptación del método *a contrario* de Desolneux, Moisan y Morel.

1.2. *Line-support regions*

El primer paso de LSD es el dividir la imagen en regiones conexas de píxeles con igual orientación, a menos de cierta tolerancia τ, llamadas *line-support regions*. El método para realizar tal división es del tipo “región creciente”; cada región comienza por un píxel y cierto ángulo asociado, que en este caso coincide con el de este primer píxel. Luego, se testean sus ocho vecinos y los que cuenten con un ángulo similar al de la región son incluídos en la misma. En cada iteración el

ángulo asociado a la región es calculado como el promedio de las orientaciones de cada píxel dentro de la *line-support region*; la iteración termina cuando ya no se pueden agregar más píxeles a esta.

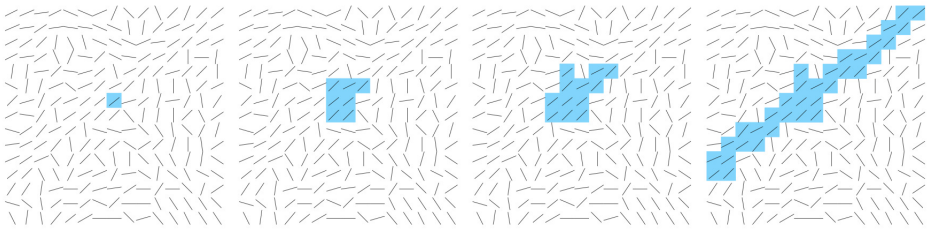

Figura 1.1: Proceso de crecimiento de una región. El ángulo asociado cada píxel de la imagen está representado por los pequeños segmentos y los píxeles coloreados representan la formación de la región. Fuente [7].

Los píxels agregados a una región son marcados de manera que no vuelvan a ser testeados. Para mejorar el desempeño del algoritmo, las regiones comienzan a evaluarse por los píxeles con gradientes de mayor amplitud ya que estos representan mejor los bordes.
Existen algunos casos puntuales en los que el proceso de búsqueda de *line-support regions* puede arrojar errores. Por ejemplo, cuando se tienen dos segmentos que se juntan y que son colineales a no ser por la tolerancia τ descripta anteriormente, se detectarán ambos segmentos como uno solo; ver figura 1.2. Este potencial problema es heredado del algoritmo de Burns, Hanson y Riseman.

Figura 1.2: Potencial problema heredado del algoritmo de Burns, Hanson y Riseman. Izq.: Imagen original. Ctro.: Segmento detectado. Der.: Segmentos que deberían haberse detectado. Fuente [7].

Sin embargo, LSD plantea un método para ahorraste este tipo de problemas. Durante el proceso de crecimiento de las regiones, también se realiza la aproximación rectangular a dicha región (paso (2) de los tres definidos anteriormente); y si menos cierto porcentaje umbral de los píxeles dentro del rectángulo corresponden a la *line-support region*, lo que se tiene no es un segmento. Se detiene entonces el crecimiento de la región.

1.3. Aproximación de las regiones por rectángulos

Cada *line-support region* debe ser asociada a un segmento. Cada segmento será determinado por su centro, su dirección, su anchura y su longitud. A diferencia de lo que pudiése dictar la intuición, la dirección asociada al segmento no se corresponde con la asociada a la región (el promedio de las direcciones de cada uno de los píxeles). Sin embargo, se elige el centro del segmento como el centro de masa de la región y su dirección como el eje de inercia principal de la misma; la magnitud del gradiente asociado a cada píxel hace las veces de masa. La idea detrás de este método es que los píxeles con un gradiente mayor en módulo, se corresponden mejor con la percepción de un borde. La anchura y la longitud del segmento son elegidos de manera de cubir el 99 % de la masa de la región.

Figura 1.3: Búsqueda del segmento que mejor aproxime cada *line-support region*: aproximación de una región por un rectángulo. Izq.: Imagen original. Ctro.: Una de las regiones computadas. Der.: Aproximación rectangular que cubre el 99 % de la masa de la región. Fuente [7].

1.4. Validación de segmentos

La validación de los segmentos previamente detectados se plantea como un método de test de hipótesis. Se utiliza un modelo *a contrario*: dada una imagen de ruido blanco y Gaussiano, se sabe que cualquier tipo de estructura detectada sobre la misma será casual. En rigor, se sabe que para cualquier imagen de este tipo, su *level-line orientation field* toma, para cada píxel, valores independientes y uniformemente distribuídos entre $[0, 2\pi]$. Dado entonces un segmento en la imagen analizada, se estudia la probabilidad de que dicha detección se dé en la imagen de ruido, y si ésta es lo suficientemente baja, el segmento se considerará válido, de lo contrario se considerará que se esta bajo la hipótesis H_0: un conjunto aleatorio de píxeles que casualmente se alinearon de manera de detectar un segmento.
Para estudiar la probabilidad de ocurrencia de una cierta detección en la imagen de ruido, se deben tomar en cuenta todos los rectángulos potenciales dentro de la misma. Dada una imagen $N \times N$, habrán N^4 orientaciones posibles para los segmentos, N^2 puntos de inicio y N^2 puntos de fin. Si se consideran N posibles valores para la anchura de los rectángulos, se obtienen N^5 posibles segmentos. Por su parte, dado cierto rectángulo r, detectado en la imagen x, se denota $k(r,x)$ a la cantidad de píxeles alineados dentro del mismo. Se define además un valor llamado *Number of False Alarms* (NFA) que está fuertemente relacionado con la probabilidad de detectar al rectángulo en cuestión en la imagen de ruido X:

$$NFA(r,x) = N^5 . P_{H_0}[k(r,X) \geq k(r,x)]$$

véase que el valor se logra al multiplicar la probabilidad de que un segmento de la imagen de ruido, de tamaño igual a r, tenga un número mayor o igual de píxeles alineados que éste, por la cantidad potencial de segmentos N^5. Cuanto menor sea el número NFA, más significativo será el segmento detectado r; pues tendrá una probabilidad de aparición menor en una imagen sin estructuras. De esta manera, se descartará H_0, o lo que es lo mismo, se aceptará el segmento detectado como válido, si y sólo si:

$$NFA(r) \leq \varepsilon$$

donde empíricamente $\varepsilon = 1$ para todos los casos.
Si se toma en cuenta que cada píxel de la imagen ruidosa toma un valor independiente de los demás, se concluye que también lo harán su gradiente y su *level-line orientation field*. De esta menera, dada una orientación aleatoria cualquiera, la probabilidad de que uno de los píxeles de la imagen cuente

con dicha orientación, a menos de la ya mencionada tolerancia τ, será:

$$p = \frac{\tau}{\pi}$$

además, se puede modelar la probabilidad de que cierto rectángulo en la imagen ruidosa, con cualquier orientación, formado por $n(r)$ píxeles, cuente con al menos $k(r)$ de ellos alineados, como una distribución binomial:

$$P_{H_0}[k(r,X) \geq k(r,x)] = B(n(r),k(r),p).$$

Finalmente, el valor *Number of False Alarms* será calculado para cada segmento detectado en la imgen analizada de la sigiuente manera:

$$NFA(r,x) = N^5.B(n(r),k(r),p);$$

si dicho valor es menor o igual a $\varepsilon = 1$, el segmento se tomará como válido; de lo contrario de dacartará.

1.5. Refinamiento de los candidatos

Por lo que se vió hasta el momento, la mejor aproximación rectangular a una *line-support region* es la que obtenga un valor NFA menor. Para los segmentos que no son validados, se prueban algunas variaciones a la aproximación original con el objetivo de disminuír su valor NFA y así entonces validarlos. Esta claro que este paso no es significativo para segmentos largos y bien definidos, ya que estos serán validados en la primera inspección; sin embargo, ayuda a detectar segmentos más pequeños y algo ruidosos.

Lo que se hace es probar distintos valores para la anchura del segmento y para sus posiciones laterales, ya que estas son los parámetros peor estimados en la aproximación rectangular, pero tienen un efecto muy grande a la hora de validar los segmentos. Es que un error de un píxel en el ancho de un segmento, puede agregar una gran cantidad de píxeles no alineados a este (tantos como el largo del segmento), y esto se ve reflejado en un valor mayor de NFA y puede llevar a una no detección.

Otro método para el refinamiento de los candidatos es la disminución de la tolerancia τ. Si los puntos dentro del rectángulo efectivamente corresponden a un segmento, aunque la tolerancia disminuya, se computará prácticamente misma cantidad de segmentos alineados; y con una probabilidad menor de ocurrencia ($\frac{\tau}{\pi}$), el valor NFA obtenido será menor. Los nuevos valores testeados de tolerancia son: $\frac{\tau}{2}$, $\frac{\tau}{4}$,$\frac{\tau}{8}$,$\frac{\tau}{16}$ y $\frac{\tau}{32}$. El nuevo valor NFA asociado al segmento será el menor de todos los calculados.

1.6. Optimización del algoritmo para tiempo real

Que un algoritmo de procesamiento de imágenes digitales sea temporalmente lineal significa que su tiempo de ejecución crece linealmente con el tamaño de la imagen en cuestión. Se sabe que estos algorimos son ideales para el procesamiento en tiempo real. Si bien, como se aclaró algunos párrafos atrás, LSD es temporalmente lineal, este no fue pensado para ser ejecutado en tiempo real. Así entonces, para poder aumentar la tasa de cuadros por segundo total de la aplicación, hubo que realizar algunos cambios mínimos en el código, simpre buscando que estos no sean sustantivos, con

el objetivo de alterar lo menos posible el desempeño del algoritmo. Se trabajó sobre ciertos bloques en particular.

1.6.1. Filtro Gaussiano

Antes de procesar la imagen con el algoritmo tal y como se vió en secciones anteriores, la misma es filtrada con un filtro Gaussiano. Se busca en primer lugar, disminuír el tamaño de la imagen de entrada con el objetivo de disminuír el volumen de información procesada. Además, al difuminar la imagen, se conservan únicamente los bordes más pronunciados. Para este proyecto en particular, se escogió la escala del submuestreo fija en $0,5$, un poco más adelante en la corriente sección se explicará por qué.

El filtrado de la imgen se hace en dos pasos, primero a lo ancho y luego a lo largo. Se utiliza el núceo Gaussiano normalizado de la figura 1.4.

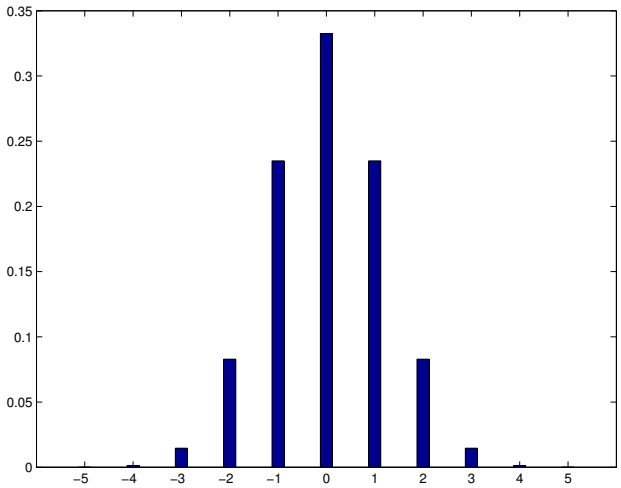

Figura 1.4: Núcleo Gaussiano utilizado por LSD. $\sigma = 1,2$.

De esta manera, se crea una imagen auxiliar vacía y escalada en x pero no en y, y se recorre asignándole a cada píxel en x su valor correspondiente, obtenido del promedio del píxel $\frac{x}{escala}$ en la imagen original y sus vecinos, todos ponderados por el núcleo Gaussiano centrado en $\frac{x}{escala}$. Luego se crea otra imagen, pero esta vez escalada tanto en x como en y, y se recorre asignándole a cada píxel en y su valor correspondiente, obtenido del promedio del píxel $\frac{y}{escala}$ en la imagen auxiliar y sus vecinos, todos ponderados por el núcleo Gaussiano centrado en $\frac{y}{escala}$. En la figura 1.5 se muestra la relación entre las imágenes.

Véase que cuando en el submuestreo $\frac{1}{escala}$ no es un entero, el centro del núcleo Gaussiano no siempre debe caer justo sobre un píxel en particular en la imagen original, sino que debe hacerlo entre dos de ellos. Lo que se hace entonces es mover $\pm 0,5$ píxeles al centro del núcleo en cada asignación de los píxeles en las imágenes escaladas; de manera de que la ponderación en el promediado de los píxeles de la imagen original (y luego la auxiliar) sea la debida. Aunque esta operación le agrega precisión al algoritmo, también le agrega un gran costo computacional, ya que lo que se

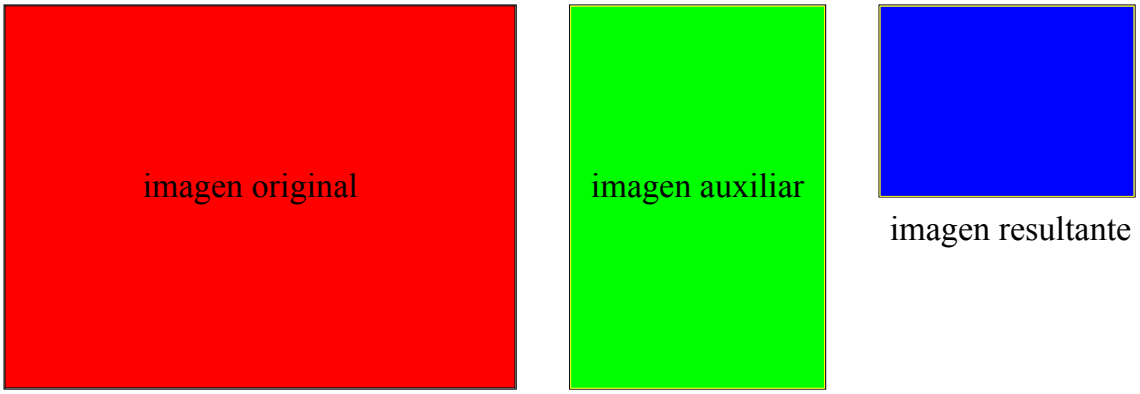

Figura 1.5: Relación entre las imágenes en consideradas en el filtro Gaussiano. Escala: $0,5$.

hace es crear un nuevo núcleo Gaussiano en cada caso. En particular, para una imagen escalada de 240×180 píxeles (dimensiones efectivamente utilizadas en este proyecto), debido al filtrado en dos pasos, el núcleo Gaussiano se crea y se destruye $86400 + 43200 = 129600$ veces.

Se decidió redondear la escala de submuestreo en $0,5$, ya que los valores utlizados empíricamente hasta el momento rondaban este valor, y se concluyó que para dicha escala, el núcleo Gaussiano debía permanecer constante, siempre centrado en su sexta muestra (ver figura 1.4); por lo que se lo quitó de la iteración y actualmente se crea una sola vez al ingresar la imagen al filtro. Es importante destacar que esta optimización es transparente para el algoritmo si y sólo si $\frac{1}{escala} = n$, donde n es un entero.

Otro cambio que se le realizó al filtrado Gaussiano fue la supresión de las condiciones de borde. Cuando se filtra cualquier imagen con un filtro con memoria, algo importante a tener en cuenta son las condiciones de borde, ya que para el procesamiento de los extremos de la imagen, estos filtros requieren de píxeles que están fuera de sus límites. Algunas de las soluciones a este problema son periodizar la imagen, simetrizarla o hasta asumir el valor 0 para los píxeles que estén fuera de esta. La opción escogida por LSD es la simetrización. Demás está decir que este proceso requeire de cierto costo computacional extra, por lo que se lo decidió suprimir. Actualmente, la imagen escalada no es computada en sus píxeles terminales; estos son 3 al inicio de cada línea o columna y 2 al final de cada una de ellas, irrelevantes en el tamaño total de la imagen. Ver figura 1.6.

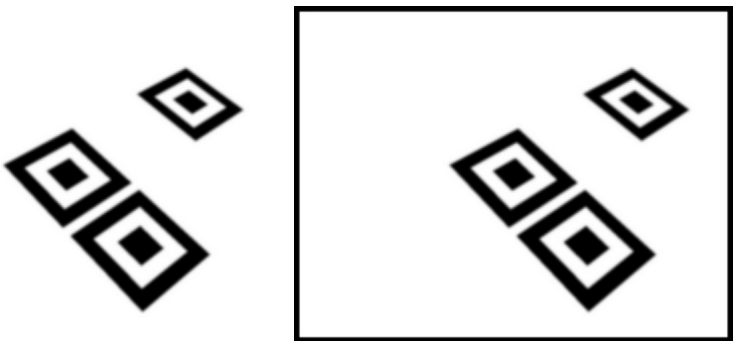

Figura 1.6: Imagen artificial del marcador trasladado y rotado, filtrada con el filtro Gaussiano. Izq.: Filtro Original. Der.: Filtro sin las condiciones de borde.

1.6.2. *Level-line angles*

La función *ll_angles* es quien calcula el gradiente de la imagen previamente filtrada para luego obtener el llamado *level-line orientation field*, en donde más tarde se hallarán los candidatos a segmentos. Lo que se hizo en esta función fué limitar el cálculo del grandiente a los píxeles donde la imagen escalada haya sido efectivamente computada. De esta manera se ahorra procesamiento innecesario, además de no detectarse las líneas negras en el contorno de la imagen (figura 1.6), que de no ser así se detectarían.

1.6.3. Refinamiento y mejora de los candidatos

Se vió en la explicación del algoritmo el problema de que si hubiesen dos o más segmentos que formen entre ellos ángulos menores o iguales al valor umbral τ, estos serían detectados como uno único, heredado del algoritmo de Burns, Hanson y Riseman; y se explicó cómo, mediante un refinamiento de los segmentos, LSD soluciona este problema. Se vió además que luego de la validación o no de los segmentos previamente detectados, se realiza una mejora de los mismos para intentar que los no validados a causa de una mala estimación rectangular, sí puedan serlo.

Como en este proyecto en particular se trabaja con marcadores formados por cuadrados concentricos, de bordes bien marcados y que forman ángulos rectos entre sí, el refinamiento y la mejora de los candidatos no es algo que afecte la detección de los mismos; y por consiguiente se suprimieron ambos bloques. Como era de esperarse, dichas supresiones no significaron un cambio considerable en el algoritmo desde el punto de vista del desempeõ ni del tiempo de ejecución cuando tan sólo se enfoca al marcador. Sin embargo, si las imágenes capturadas cuentan con muchos segmentos (imágenes naturales genéricas), se ve que la detección de los mismos es menos precisa que la del algoritmo original, pero que los tiempos de procesamiento son notablemente inferioes.

1.6.4. Algorirmo en precisión simple

Originalmente, LSD fue implementado en precisión doble o *double* (64 bits por valor). Sin embargo, el *ipad 2* (dispositivo para el cual se optimizó el algoritmo), cuenta con un procesador *ARM Cortex-A9*, cuyo bus de datos es de 32 bits. Se decidió entonces probar cambiar al algoritmo a precisión simple o *float* (32 bits por valor) y los resultados fueron realmente buenos. No sólo el algoritmo bajó su tiempo de ejecución, sino que además no existen cambios notorios en el desempeño del mismo.

1.6.5. Resultados

1.6.5.1. Filtro Gaussiano

Figura 1.7: Imagen sintética del marcador trasladado y rotado.

Se analizaron los tiempos promedio para la ejecución del filtro Gaussiano original y del optimizado, ambos con precisión doble y simple. Las imagen de prueba fue la de la figura 1.7; sépase que por cómo es el algoritmo, el contenido de la imagen en independiente del tiempo de procesamiento en cualquiera de los casos. Los valores relevantes del experimento se muestran en las tablas 1.1 y 1.2:

- **Precisión doble (*double*)**

	Filtro original	Filtro optimizado
Tamaño de imagen de entrada	480×360	480×360
Escala	$0,5$	$0,5$
Tamaño de imagen de salida	240×180	240×180
Segmentos detectados	36	36
Tiempo medio de procesamiento	**36ms**	**29ms**

Tabla 1.1: Comparación entre los tiempos de ejecución del filtro Gaussiano optimizado y el original. Ambos comprecisión doble.

- **Precisión simple (*float*)**

	Filtro original	Filtro optimizado
Tamaño de imagen de entrada	480×360	480×360
Escala	$0,5$	$0,5$
Tamaño de imagen de salida	240×180	240×180
Segmentos detectados	36	36
Tiempo medio de procesamiento	**28ms**	**20ms**

Tabla 1.2: Comparación entre los tiempos de ejecución del filtro Gaussiano optimizado y el original. Ambos comprecisión simple.

1.6.5.2. *Line Segment Detection*

Se analizaron los tiempos conjuntos para la ejecucición de LSD más el filtro Gaussiano, los originales y los optimizados, ambos con precisión doble y simple. Se probaron ambos bloques juntos

Figura 1.8: Imagen *zebras.png*.

ya que el algoritmo original está implementado con éstos integrados. Las imágenes de prueba fueron la del marcador sintético (figura 1.7) y *zebras.png* mostrada en la figura 1.8. Los valores relevantes de los experimentos se muestran en las tablas 1.3, 1.4, 1.5 y 1.6.

- **Precisión doble (*double*)**

	Algoritmo original	Algoritmo optimizado
Imagen urilizada	marcador sintético	marcador sintético
Tamaño de imagen de entrada	480×360	480×360
Escala	$0,5$	$0,5$
Tamaño de imagen de salida	240×180	240×180
Segmentos detectados	36	36
Tiempo medio de procesamiento	**55,4ms**	**48ms**

Tabla 1.3: Comparación entre los tiempos de ejecución del filtro Gaussiano más LSD optimizados y los originales, para la imagen 1.7. En todos los casos con comprecisión doble.

	Algoritmo original	Algoritmo optimizado
Imagen urilizada	*zebras.png*	*zebras.png*
Tamaño de imagen de entrada	480×360	480×360
Escala	$0,5$	$0,5$
Tamaño de imagen de salida	240×180	240×180
Segmentos detectados	251	179
Tiempo medio de procesamiento	**179,7ms**	**94,4ms**

Tabla 1.4: Comparación entre los tiempos de ejecución del filtro Gaussiano más LSD optimizados y los originales, para la imagen 1.8. En todos los casos con comprecisión doble.

- **Precisión simple (*float*)**

	Algoritmo original	Algoritmo optimizado
Imagen urilizada	marcador sintético	marcador sintético
Tamaño de imagen de entrada	480×360	480×360
Escala	$0,5$	$0,5$
Tamaño de imagen de salida	240×180	240×180
Segmentos detectados	36	36
Tiempo medio de procesamiento	**47,8ms**	**38,8ms**

Tabla 1.5: Comparación entre los tiempos de ejecución del filtro Gaussiano más LSD optimizados y los originales, para la imagen 1.7. En todos los casos con comprecisión simple.

	Algoritmo original	Algoritmo optimizado
Imagen urilizada	*zebras.png*	*zebras.png*
Tamaño de imagen de entrada	480×360	480×360
Escala	$0,5$	$0,5$
Tamaño de imagen de salida	240×180	240×180
Segmentos detectados	252	182
Tiempo medio de procesamiento	**189,8ms**	**90,8ms**

Tabla 1.6: Comparación entre los tiempos de ejecución del filtro Gaussiano más LSD optimizados y los originales, para la imagen 1.8. En todos los casos con comprecisión simple.

Bibliografía

[1] J. García Ocón. Autocalibración y sincronización de múltiples cámaras plz. 2007.

[2] B. Furht. *The Handbook of Augmented Reality*. 2011.

[3] C. Avellone and G. Capdehourat. Posicionamiento indoor con señales wifi. 2010.

[4] Philip David, Daniel Dementhon, Ramani Duraiswami, and Hanan Samet. Simultaneous pose and correspondence determination using line features. pages 424–431, 2003.

[5] Philip David, Daniel Dementhon, Ramani Duraiswami, and Hanan Samet. Softposit: Simultaneous pose and correspondence determination. pages 424–431, 2002.

[6] Daniel F. DeMenthon and Larry S. Davis. Model-based object pose in 25 lines of code. *International Journal of Computer Vision*, 15:123–141, 1995.

[7] R. Grompone von Gioi, J. Jakubowicz, J. M. Morel, and G. Randall. Lsd: A fast line segment detector with a false detection control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(4):722–732, April 2010.

[8] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[9] V. Lepetit and P. Fua. Monocular model-based 3d tracking of rigid objects: A survey. *Foundations and Trends in Computer Graphics and Vision*, 1(1):1–89, 2005.