CAPÍTULO 1

LSD: "Line Segment Detection"

1.1. Introducción

LSD es un algoritmo de detección de segmentos publicado recientemente [18]. Es temporalmente lineal, tiene precisión inferior a un píxel y no requiere de un ajuste 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, lo cual puede ser detectado mediante el módulo del gradiente de la misma.

Se genera en primer lugar, 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, a menos de 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 [5], y el punto (3) es una adaptación del método *a contrario* de Desolneux, Moisan y Morel [12].

En el presente capítulo se estudiará a fondo el algoritmo y se presentarán y justificarán algunos cambios que hubo que hacerle a la imlpementación del mismo con la que se contaba, versión 1.6 descargada de [42], para mejorar su desempeño en el tiempo real.

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 "*region growing*"; 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 la misma.

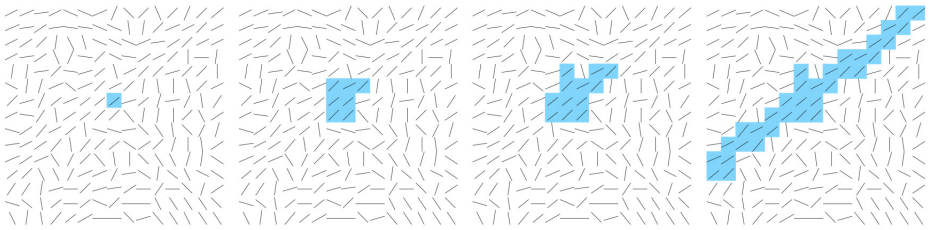

Figura 1.1: Proceso de crecimiento de una región. El ángulo asociado cada píxel de la imagen está representado por los pequeños segmentos y los píxeles coloreados representan la formación de la región. Tomada de [18].

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. Tomada de [18].

Sin embargo, LSD plantea un método para solucionar 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 de 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

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. Tomada de [18].

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 pudiese resultar intuitivo, 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, tienen una mayor probabilidad de corresponder a un borde. La anchura y la longitud del segmento son elegidos de manera de cubir el 99 % de la masa de la región.

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*. El término *a contrario* viene del latín y significa "al revés" o "de forma opuesta". En procesamiento de imágenes, el principio para la detección *a contrario* define, en primer lugar, un modelo llamado "*a priori*" para el caso genérico en el que no haya nada que detectar. Entonces la detección de un evento en particular sólo se dará cuando la cantidad de ocurrencias de dicho evento en el modelo *a priori* sea lo suficientemente baja. Nótese la aparición de cierto valor umbral a ajustar.

Para el caso de LSD, 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 manera, 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. Estos algoritmos son los mejores para el procesamiento de imágenes en tiempo real. Si bien, como se explicó con anterioridad, los autores de LSD afirman que este es temporalmente lineal; la implementación con la que se cuenta no fue pensada para ser ejecutada 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 alteren 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, disminuir el tamaño de la imagen de entrada con el objetivo de disminuir 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.

Como la función Gaussiana 2D es separable, el filtrado de la imagen se hace en dos pasos, primero a lo ancho y luego a lo largo. Se utiliza el núceo Gaussiano de una dimensión normalizado de la Figura 1.4.

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

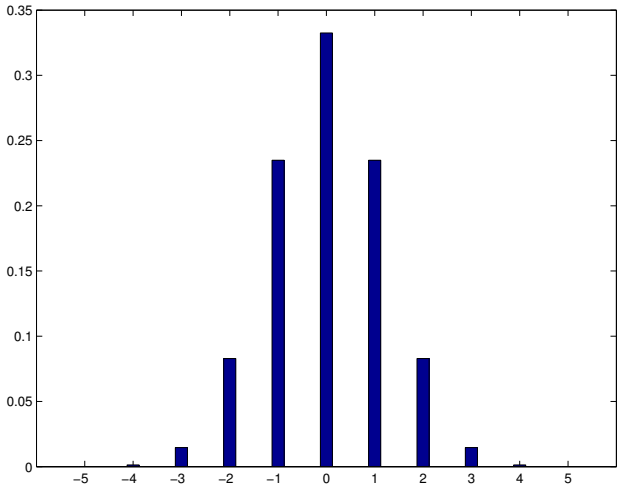

Figura 1.4: Núcleo Gaussiano utilizado por LSD. $\sigma = 1,2$.

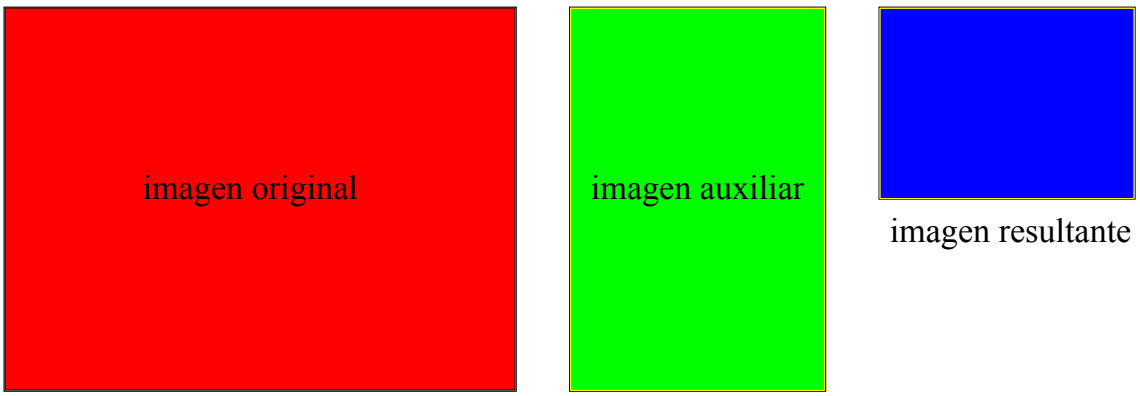

Figura 1.5: Relación entre las imágenes en consideradas en el filtro Gaussiano. Escala: $0,5$.

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 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. Este costo computacional extra se debe a que el algoritmo encargado del filtrado debe estar en cada bucle preguntándose si es necesario contar con el valor de algún píxel fuera de los límites de la imagen, y en ese caso asignarle a dicho píxel el valor de su correspondiente simétrico, con eje de simetría el borde de la imagen más próximo. 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 y también, por ser un filtro FIR ("Finite Impulse Response"), en el resultado del filtrado en general. Ver Figura 1.6.

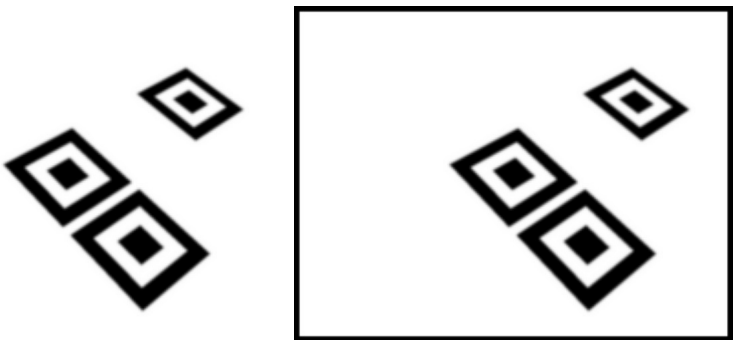

Figura 1.6: Imagen artificial del marcador trasladado y rotado, filtrada con el filtro Gaussiano. Izq.: Filtro Original. Der.: Filtro sin las condiciones de borde.

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 inferiores.

1.6.4. Algoritmo en precisión simple

Originalmente, LSD fue implementado en precisión doble o *double* (en general 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. La imagen de prueba fue la de la Figura 1.7; sépase que por cómo es el algoritmo, el contenido de la imagen es independiente del tiempo de procesamiento en cualquiera de los casos, por lo que basta con una única imagen de prueba para sacar conslusiones respecto del desempeño del mismo. 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 con precisión 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 con precisión simple.

1.6.5.2. *Line Segment Detection*

Figura 1.8: Imagen *zebras.png*.

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

	Algoritmo original	Algoritmo optimizado
Imagen utilizada	*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 precisión doble.

- **Precisión simple (*float*)**

	Algoritmo original	Algoritmo optimizado
Imagen utilizada	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 precisión simple.

	Algoritmo original	Algoritmo optimizado
Imagen utilizada	*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 precisión simple.

1.7. Conslusión

En el presente capítulo se vió en detalle LSD, un algoritmo de detección de segmentos en imágenes que puede ser considerado el estado del arte en su rubro. Luego se afirmó que, si bien sus autores sostienen que el algoritmo es temporalmente lineal, lo que haría viable su uso en tiempo real; la implementación disponible del mismo, realizada por los propios autores, no está optimizada para tal caso y por eso hubo que hacerle algunos prqueños cambios al código. Dichos cambios lograron mejoras importantes en cuanto al tiempo de procesamiento, manteniendo prácticamente invariado su desempeño.

Los resultados expuestos en las Tablas 1.1 y 1.2 pueden interpretarse como que la decisión de cambiar la precisión de la implementación del algoritmo de *double* a *float* fue una idea acertada. Por

su parte, la lectura de las Tablas 1.3, 1.4, 1.5 y 1.6 sugiere que las mejoras en los tiempos de LSD optimizado para el tiempo real, respecto del original, son del órden de:

30 % para imágenes con pocos segmentos;
50 % para imágenes naturales genéricas, con muchos segmentos.

Cabe aclarar sin embargo, que si bien los resultados cualitativos[1] sugieren resultados similares, los resultados cuantitativos fueron logrados con tan sólo las dos imágenes presentadas anteriormente.

Finalmente, en las Figuras 1.9 y 1.10 se muestan los resultados luego de haber procesado a las Figuras 1.7 y 1.8 con LSD, primero con la implementación original y luego con la optimizada. Se puede ver, en primer lugar, la gran diferencia que existe entre la cantidad de segmentos detectados en una y otra imagen. Además, se concluye que el desempeño de ambas implementaciones es muy similar para el caso de la imagen *zebras.png* e idéntico para el caso del marcador.

Figura 1.9: Resultado de procesar a la Figura 1.7 con LSD. Izq.: Implementación original. Der.: Implementación optimizada.

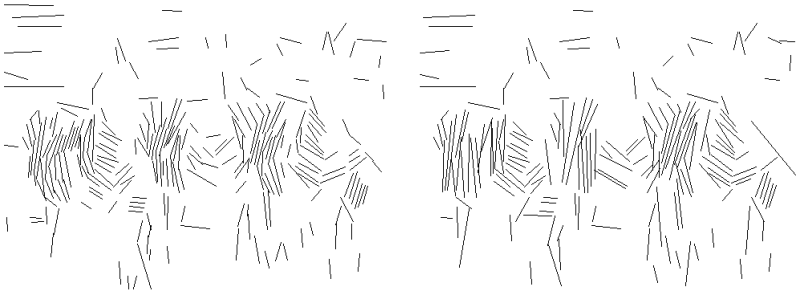

Figura 1.10: Resultado de procesar a la Figura 1.8 con LSD. Izq.: Implementación original. Der.: Implementación optimizada.

[1] Se entiende por resultados cualitativos a la ejecución de LSD en tiempo real en el *iPad*, imposibles de ilustrar en el presente texto. Ver Sección **??**.

Bibliografía

[1] A.Etemadi. Robust segmentation of edge data. In *Proceedings of the 4th international conference on Image Processing and its applications*, 1992.

[2] Y. I. Abdel-Aziz and H. M. Karara. Direct linear transformation from comparator coordinates into object space coordinates in close-range photogrammetry. In *Proceedings of the Symposium on Close-Range photogrammetry*, volume 1, pages 1–18, 1971.

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

[4] Jean-Yves Bouguet. Camera calibration toolbox for matlab. `http://www.vision.caltech.edu/bouguetj/calib_doc/`, November 2012.

[5] J. Brian Burns, Allen R. Hanson, and Edward M. Riseman. Extracting straight lines. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8:425–455, 1986.

[6] F. John Canny. A computational approach to edge detection. *IEEE-PAMI*, 8(6):679–698, 1986.

[7] Stuart Caunt. Isgl3d homepage. `http://www.isgl3d.com`, nov 2012.

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

[9] Philip David, Daniel DeMenthon, Ramani Duraiswami, and Hanan Samet. Simultaneous pose and correspondence determination using line feature. In *CVPR (2)*, pages 424–431, 2003.

[10] 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.

[11] Daniel DeMenthon Denis Oberkampf. Posit for coplanar points. `http://www.cfar.umd.edu/~daniel/Site_2/Code.html`, 1996 - 2004.

[12] Agnès Desolneux, Lionel Moisan, and Jean-Michel Morel. Meaningful alignments. *International Journal of Computer Vision*, 40(1):7–23, 2000.

[13] A. Etemadi, J-P. Schmidt, G.Matas, J.Illingworth, and J. Kittler. Low-level grouping of straight-line segments. In Peter Mowforth, editor, *Processings of the British Machine Vision Conference*. Springer-Verlag, 1991.

[14] Mark Fiala. Artag revision 1, a fiducial marker system using digital techniques. `http://www.artag.net/`, November 2004.

[15] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, jun 1981.

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

[17] Martin Giupponi. Kalman robusto aplicado en segmentación de videos con texturas dinámicas. Tratamiento Estadístico de Señales. Facultad de Ingeniería, Universidad de la República, 2009.

[18] Rafael Grompone von Gioi, Jérémie Jakubowicz, J.-M. Morel, and Gregory Randall. Lsd: a line segment detector. *Image Processing Online*, mar 2012.

[19] Rafael Grompone von Gioi, Jéremie Jakubowicz, J.-M. Morel, and Gregory Randall. On straight line segment detection. *Journal of Mathematical Imaging and Vision*, 2008.

[20] Robert M. Haralick, Chung-Nan Lee, Karsten Ottenberg, and Michael Nölle. Review and analysis of solutions of the three point perspective pose estimation problem. *Int. J. Comput. Vision*, 13(3):331–356, dec 1994.

[21] Chris Harris and Mike Stephens. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*, pages 147–151, 1988.

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

[23] Monson H. Hayes. *Statistical Digital Signal Processing and Modeling*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1996.

[24] Jane Heikkilä and Olli Silvén. A four-step camera calibration procedure with implicit image correction. In *1997 Conference on Computer Vision and Pattern Recognition (CVPR 97), June 17-19, 1997, San Juan, Puerto Rico*, page 1106. IEEE Computer Society, 1997.

[25] Martin Hirzner. Marker detection for augmented reality applicactions. October 2008.

[26] Dr. Hirokazu Kato. Artoolkit: a software library for building augmented reality (ar) applications. `http://www.hitl.washington.edu/artoolkit/`.

[27] 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.

[28] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, nov 2004.

[29] David G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ICCV '99, pages 1150–, Washington, DC, USA, 1999. IEEE Computer Society.

[30] Denis Oberkampf, Daniel F. DeMenthon, and Larry S. Davis. Iterative pose estimation using coplanar feature points. *Comput. Vis. Image Underst.*, 63(3):495–511, may 1996.

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

[32] Matias Tailanian and Santiago Paternain. Autoposicionamiento 3d. `http://sites.google.com/site/autoposicionamiento3d/`, Julio 2011.

[33] R.Y. Tsai. An efficient and accurate camera calibration technique for 3d machine vision. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 364–374, 1986.

[34] Daniel Wagner and Dieter Schmalstieg. Artoolkitplus for pose tracking on mobile devices. 2007.

[35] Zhengyou Zhang. Flexible camera calibration by viewing a plane from unknown orientations. In *ICCV*, pages 666–673, 1999.

[36] Helmut Zollner and Robert Sablatnig. Comparision of methods for geometric camera calibration using planar calibration targets. In W. Burger and J. Scharinger, editors, *Digital Imaging in Media and Education, Proc. of the 28th Workshop of the Austrian Association for Pattern Recognition (OAGM/AAPR)*, volume 179, pages 237–244. Schriftenreihe der OCG, 2004.

[37] Cihan Topal and Cuneyt Akinlar. Edge drawing: A combined real-time edge and segment detector. *Journal of Visual Communication and Image Representation*, 23(6):862 – 872, 2012.

[38] Cuneyt Akinlar and Cihan Topal. Edlines: A real-time line segment detector with a false detection control. *Pattern Recognition Letters*, 32(13):1633 – 1642, 2011.

[39] Blender homepage. `http://www.blender.org/`.

[40] Blender 2.6 python manual. `http://wiki.blender.org/index.php/Doc:2.6/Manual/Extensions/Python`.

[41] Image processing on line. canny demo. `http://dev.ipol.im/~coco/ipol_demo/canny/`.

[42] Image processing on line. lsd: a line segment detector. `http://www.ipol.im/pub/art/2012/gjmr-lsd/`, nov 2012.

[43] Vlfeat homepage. `http://www.vlfeat.org`, nov 2012.

[44] iphone 4 support specification. `http://support.apple.com/kb/SP587`, nov 2012.

[45] iphone 4s support specification. `http://support.apple.com/kb/SP643`, nov 2012.

[46] ipod touch support specification. `http://support.apple.com/kb/SP594`, nov 2012.

[47] ipad 2 support specification. `http://support.apple.com/kb/SP622`, nov 2012.

[48] Apple platform notes. `http://developer.apple.com/library/ios/#documentation/3DDrawing/Conceptual/OpenGLES_ProgrammingGuide/OpenGLESPlatforms/OpenGLESPlatforms.html`, nov 2012.

[49] Peter Thoman. *Microcontroller and System-on-a-chip*. University of Innsbruck, second edition, 2009.

[50] Arm cortex-a processors. `http://www.arm.com/products/processors/cortex-a/index.php`, nov 2012.

[51] Gpu benchmark. `http://www.anandtech.com/show/4216/apple-ipad-2-gpu-performance-explored-powervr-sgx543mp2-benchmarked`, nov 2012.

[52] Powervr graphics technology. `http://www.imgtec.com/powervr/powervr-graphics-technology.asp`, nov 2012.

[53] The objective-c programming language. http://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/ObjectiveC/Introduction/introObjectiveC.html, nov 2012.

[54] ios technology overview. http://developer.apple.com/library/ios/#documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html#//apple_ref/doc/uid/TP40007898-CH1-SW1, nov 2012.

[55] Av foundation programming guide. http://developer.apple.com/library/ios/#DOCUMENTATION/AudioVideo/Conceptual/AVFoundationPG/Articles/00_Introduction.html#//apple_ref/doc/uid/TP40010188, nov 2012.

[56] Advanced memory management programming guide. http://developer.apple.com/library/ios/#documentation/Cocoa/Conceptual/MemoryMgmt/Articles/MemoryMgmt.html#//apple_ref/doc/uid/10000011i, nov 2012.

[57] Transitioning to arc release notes. http://developer.apple.com/library/ios/#releasenotes/ObjectiveC/RN-TransitioningToARC/Introduction/Introduction.html#//apple_ref/doc/uid/TP40011226, nov 2012.

[58] Instruments user guide. https://developer.apple.com/library/ios/#documentation/DeveloperTools/Conceptual/InstrumentsUserGuide/Introduction/Introduction.html#//apple_ref/doc/uid/TP40004652, nov 2012.

[59] Metaio homepage. http://www.metaio.com/, 2012.

[60] Vuforia homepage. http://www.qualcomm.com/solutions/augmented-reality, 2012.

[61] String homepage. http://www.poweredbystring.com/, 2012.

[62] Layar homepage. http://www.layar.com/, 2012.

[63] Aurasma homepage. http://www.aurasma.com/, 2012.

[64] Mathworks documentation center. hough lines. http://www.mathworks.com/help/images/ref/houghlines.html, 2012.

[65] Cse 576 computer vision. software. object recognition toolkit (ort). http://www.cs.washington.edu/education/courses/cse576/07sp/software/index.html, 2012.

[66] Repositorio svn para descarga de object recognition toolkit (ort 2.3). http://www.cs.washington.edu/education/courses/cse576/07sp/software/index.html, 2012.

[67] Edlines: A real-time line segment detector with a false detection control. homepage. http://ceng.anadolu.edu.tr/cv/EDLines/, 2012.