Índice general

CAPÍTULO 1

Hardware y Software

1.1. Introducción

Introducción

1.2. Software de procesamiento de imágenes

Software de procesamiento de imágenes

1.3. Dispositivos móviles

Al trabajar con Apple se cuenta con la ventaja de contar con pocas variantes en cuanto al Hardware utilizado. Básicamente existen tres tipos de dispositivos en los que se pueden desarrollar: iPhone, iPad y iPod Touch. Para cada variante de plataforma existen distintos modelos que hacen que algunas características importantes como capacidad de procesamiento, resolución de cámara o tama?o de la pantalla entre otras puedan verse afectadas. A continuación se relata el surgimiento de cada uno de lo dispositivos al mercado y se describen brevemente las principales características.

1.3.1. iPhone

Sin dudas el iPhone fue uno de los saltos más grandes en el mundo tecnológico en los últimos a?os. Logró llenar el hueco que los PDAs de la década de los 90 no habían sabido completar y comenzó a desplazar al invento que revolucionó el mercado de los contenidos de música, el iPod. Gracias a su pantalla táctil capacitiva de alta sensibilidad logró reunir todas las funcionalidades agregando solamente un gran botón y algunos extra para controlar volumen o desbloquar el dispositivo.
La primera generación del iPhone fue lanzada por Apple en Junio de 2007 en Estados Unidos, luego de una gran inversión de la operadora ATT que exigía exclusividad de venta dentro de dicho país durante los siguientes cuatro a?os. La misma soportaba tecnología GSM cuatribanda y se lanzó en dos variantes de 4GB y 8GB de ROM. El segundo modelo lanzó como novedad el soporte de tecnología 3G cuatribanda y GPS asistido. Luego le siguieron el iPhone 3GS, 4, 4S y el 5, siendo este último, la sexta y última generación disponible al momento de la redacción de este trabajo.
Las dimensiones del iPhone 5 son de 58.6 x 123.8 x 7.6 millimetres, resolución de pantalla de 640 x 1136, tiene una velocidad de reloj en la CPU de 1200MHz, RAM de 1GB y la ROM varía según la variante (16GB, 30GB o 64GB).

1.3.2. iPad

Esta línea de dispositivos es la más potente en lo que respecta a capacidad de procesamiento.

1.3.3. iPod Touch

1.4. Entorno de desarrollo

Entorno de desarrollo

1.4.1. xCode y Objective-C

1.4.2. Simulador

1.4.3. Instruments

1.4.3.1. Time Profiler

1.4.3.2. Memory Leak

1.4.4. Librerías

1.4.4.1. AvFoundation

1.4.4.2. MediaPlayer

1.4.4.3. CoreMotion

1.4.4.4. Tweeter y mensajería

1.5. Herramientas

Herramientas

CAPÍTULO 2

Detección

Hola

2.1. Típos de características

Intro y mas breves definiciones?.
Bordes, esquinas, líneas, segmentos de línea, regiones (blobs).

2.2. Bordes y esquinas

2.2.1. Detector de bordes de Canny

2.2.2. Detector de bordes y esquinas de Harris

2.2.3. SUSAN Y FAST

2.3. Líneas y segmentos de línea

2.3.1. Detector de líneas de Hough

2.3.2. Detector de segmentos de línea: LSD

2.4. Regiones y puntos de interés

2.4.1. FAST

2.4.2. Blobs

2.5. Detección sin primitivas markerless

SIFT (puntero a capitulo que tiene SIFT para reconocimiento) SURF, ETC ETC.

2.6. Marcadores

Marcadores que se usan. Limitaciones

La inclusión de *marcadores*, *marcas de referencia* o *fiduciales*, en inglés *markers*, *landmarks* o *fiducials*, en la escena ayuda al problema de extracción de características y por lo tanto al problema de estimación de pose [9]. Estos por construcción son elementos que presentan una detección estable en la imagen para el tipo de característica que se desea extraer así como medidas facilmente utilizables para la estimación de la pose.

Se distinguen dos tipos de *fiduciales*. El primer tipo son los que se llaman puntos *fiduciales* por que proveen una correspondencia de puntos entre la escena y la imagen. El segundo tipo, *fiduciales planares*, se pueden obtener mediante la construcción en una geometría coplanar de una serie de *puntos fiduciales* identificables como esquinas. Un único *fiducial planar* puede contener por si solo todas las seis restricciones espaciales necesarias para definir el marco de coordenadas.

Como se explica en la sección **??** el problema de estimación de pose requiere de una serie de correspondencias $\mathbf{M}_i \leftrightarrow \mathbf{m}_i$ entre puntos 3D en la escena en coordenadas del mundo y puntos en la imagen.

2.7. Marcador QR

El enfoque inicial elegido para la detección de *puntos fiduciales* para marcadores parte del trabajo de fin de curso de Matías Tailanian para el curso *Tratamiento de imágenes por computadora* de Facultad de Ingeniería, Universidad de la Republica[1]. La elección se basa principalmente en los buenos resultados obtenidos para dicho trabajo con un enfoque relativamente simple. El trabajo desarrolla, entre otras cosas, un diseño de marcador y un sistema de detección de marcadores basado en el detector de segmentos LSD[7] por su buena performance y aparente bajo costo computacional.

El marcador utilizado está basado en la estructura de detección incluida en los códigos *QR* y se muestra en la figura 2.1. Éste consiste en trés grupos idénticos de tres cuadrados concéntricos superpuestos de tal forma que los lados de cada uno de trés cuadrados son visualizables. A diferencia de los códigos *QR* la disposición de los grupos de cuadrados es distinto para evitar ambiguedades en la determinación de su posicionamiento espacial. Estas dos características son escenciales para la extracción de los *puntos fiduciales* de forma coherente, es decir, las correspondencias tienen que poder ser determinadas completamente bajo criterios razonables.

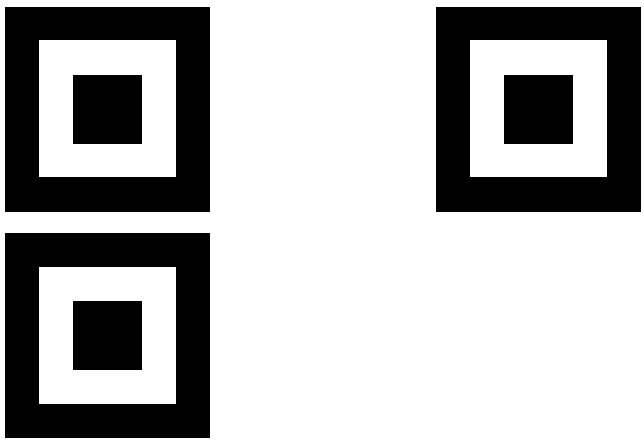

Figura 2.1: Marcador propuesto basado en la estructura de detección de códigos QR.

[1] Autoposicionamiento 3D - http://sites.google.com/site/autoposicionamiento3d/

2.7.1. Estructura del marcador

A continuación se presentan algunas definiciones de las estructuras básicas que constituyen el marcador propuesto. Estas son de utilidad para el diseño y forman un flujo natural y escalable para el desarrollo del algoritmo de determinación de correspondencias.

Los elementos mas básicos en la estructura son los *segmentos* los cuales consisten en un par de puntos en la imagen, $\mathbf{p} = (p_x, p_y)$ y $\mathbf{q} = (q_x, q_y)$. Estos *segmentos* forman lo que son los lados del *cuadrilátero*, el próximo elemento estructural del marcador.

Un *cuadrilátero* o *quadrilateral* en inglés, al que se le denomina *Ql*, está determinado por cuatro segmentos conexos y distintos entre sí. El cuadrilátero tiene dos propiedades notables; el *centro* definido como el punto medio entre sus cuatro vértices y el *perímetro* definido como la suma de el largo de sus cuatro lados. Los *vértices* de un *cuadrilátero* se determinan mediante la intersección, en sentido amplio, de dos segmentos contiguos. Es decir, si s_1 es contiguo a s_2 dadas las recta r_1 que pasa por los puntos $\mathbf{p}_1$, $\mathbf{q}_1$ del segmento s_1 y la recta r_2 que pasa por los puntos $\mathbf{p}_2$, $\mathbf{q}_2$ del segmento s_2, se determina el vértice correspondiente como la intersección $r_1 \cap r_2$.

A un *conjunto de cuadriláteros* o *quadrilateral set* se le denomina *QlSet*, se construye a partir de M cuadriláteros, que comparten un mismo centro, y se diferencian por un factor de escala, con $M > 1$. A partir de dichos cuadriláteros se construye un lista ordenada $(Ql[0], Ql[1], \ldots, Ql[M-1])$ en donde el orden viene dado por el valor de *perímetro* de cada *Ql*. Se define el *centro del grupo de cuadriláteros* como el promedio de los centros de cada *Ql* de la lista ordenada.

Finalmente el *marcador QR* está constituido por *N conjuntos de cuadriláteros* dispuestos en una geometría particular. Esta geometría permite la determinación un sistema de coordenadas; un origen y dos ejes a utilizar. Se tiene una lista ordenada $(QlSet[0], QlSet[1], \ldots, QlSet[N-1])$ en donde el orden se puede determinar mediante la geometría de los mismos o a partir de hipótesis razonables.

Un marcador proveerá un numero de $4 \times M \times N$ vértices y por lo tanto la misma cantidad de puntos fiduciales para proveer las correspondencias $\mathbf{M}_i \leftrightarrow \mathbf{m}_i$ al algoritmo de estimación de pose.

2.7.2. Diseño

En base a las estructuras previamente definidas es que se describe el diseño del marcador. Como ya se explicó se toma un marcador tipo *QR* basado en *cuadriláteros* y mas específicamente en tres conjuntos de tres cuadrados dispuestos en como se muestra en la figura 2.1.

Los tres *cuadriláteros* correspondientes a un mismo *conjunto de cuadriláteros* tienen idéntica alineación e idéntico centro. Los diferencia un factor de escala, esto es, $Ql[0]$ tiene lado l mientras que $Ql[1]$ y $Ql[2]$ tienen lado $2l$ y $3l$ respectivamente. Esto se puede ver en la figura 2.2. Adicionalmente se define un sistema de coordenadas con centro en el centro del *QlSet* y ejes definidos como **x** horizontal a la derecha e **y** vertical hacia abajo. Esta convención en las direcciones de los ejes es muy utilizada en el ambiente del *tratamiento de imágenes* para definir las direcciones de los ejes de una imagen. Definido el sistema de coordenadas de puede fijar un orden a los *vértices* v_{j_1} de cada *cuadrilátero* $Ql[j]$ como,

$$v_{j_0} = (a/2, a/2) \qquad v_{j_2} = (-a/2, -a/2)$$
$$v_{j_1} = (a/2, -a/2) \qquad v_{j_3} = (-a/2, a/2)$$

con $a = (j+1) \times l$. El orden aquí explicado se puede ver también junto con el sistema de coordenadas en la figura 2.3.

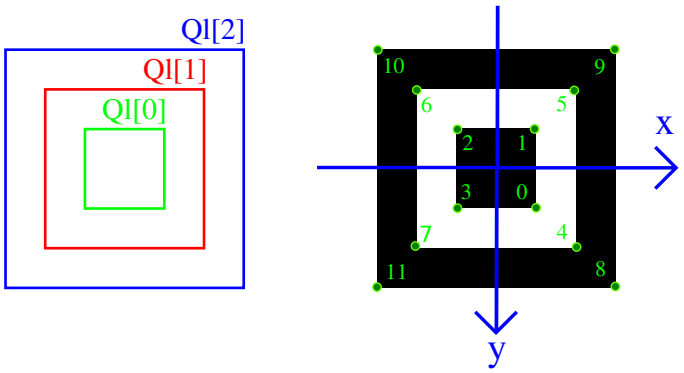

Figura 2.2: Detalle de un QlSet. Orden de los *cuadriláteros* a la izquiera. Órden de los *vértices* a la derecha.

Un detalle del marcador completo se muestra en la figura 2.3 en donde se define el conjunto i de cuadriláteros concéntricos como el $QlSet[i]$ y se definen los respectivos centros $\mathbf{c}_i$ para cada $QlSet[i]$. El sistema de coordenadas del *marcador QR* tiene centro en el centro del $QlSet[0]$ y ejes de coordenadas idénticos al definido para cada Ql. Se tiene además que los ejes de coordenadas pueden ser obtenidos mediante los vectores normalizados,

$$\mathbf{x} = \frac{\mathbf{c}_1 - \mathbf{c}_0}{||\mathbf{c}_1 - \mathbf{c}_0||} \quad \mathbf{y} = \frac{\mathbf{c}_2 - \mathbf{c}_0}{||\mathbf{c}_2 - \mathbf{c}_0||} \tag{2.1}$$

La disposición de los QlSet es tal que la distancia indicada d_{01} definida como la norma del vector entre los centros $\mathbf{c}_1$ y $\mathbf{c}_0$ es significativamente mayor que la distancia d_{02} definida como la norma del vector entre los centros $\mathbf{c}_2$ y $\mathbf{c}_1$. Esto es, $d_{01} \gg d_{02}$. Este criterio facilita la identificación de los *vértices* de los *QlSet* entre sí basados únicamente en la posición de sus centros y se explicará en la parte de determinación de correspondencias (sec.: 2.7.3.3).

2.7.2.1. Parámetros de diseño

Provisto el diseño del marcador antes descrito, quedan definidos ciertos parámetros **estructurales** que fueron de tomados fijos a lo largo del proyecto pero que podrían se variados en trabajos futuros asociados. Estos parámetros son:

- M: cantidad de *conjuntos de cuadriláteros*.
- N: cantidad de *cuadriláteros* por *conjuntos de cuadriláteros*.
- Geometría: geometría de los cuadriláteros (*Ql*).
- Disposición: disposición espacial de los *conjuntos de cuadriláteros* (*QlSet*).

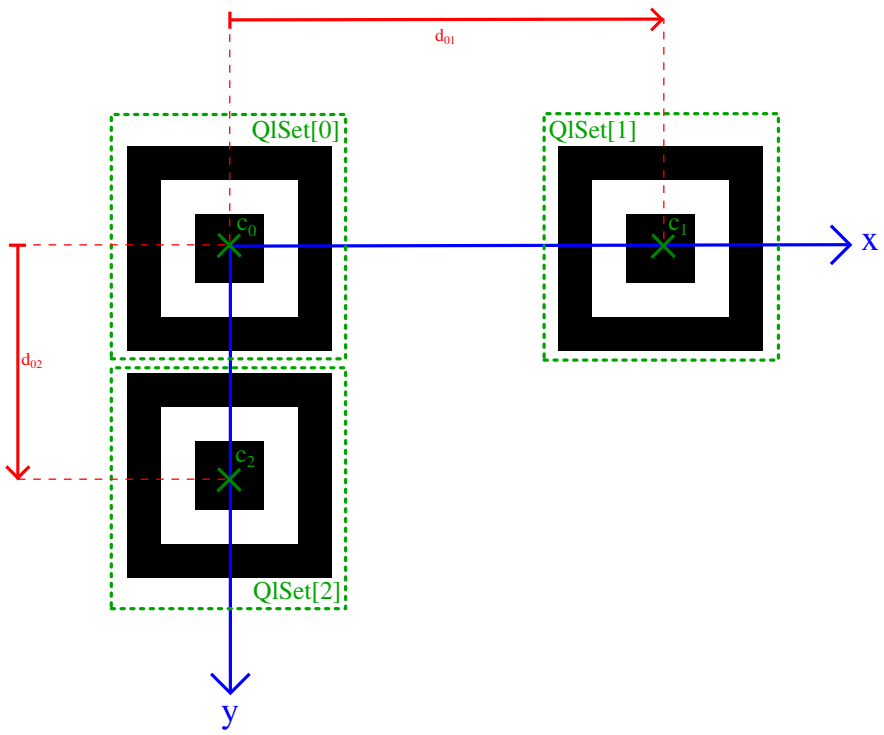

Figura 2.3: Detalle del marcador propuesto formando un sistema de coordenadas.

El criterio de elección de M y N parte del diseño los códigos QR como ya fue explicado. La detección por segmentos de línea resulta una cantidad de $3 \times QlSet$'s conteniendo $3 \times Ql$'s cada uno. Bajo esta elección de parámetros se tienen 36 *segmentos* y *vértices*. Se tienen entonces un número de puntos característicos razonable para la estimación de pose.

La elección de *cuadrados* como parámetro de geometría se basa en la necesidad de tener igual resolución en los dos ejes del marcador. De esta forma se asegura una distancia límite en donde, en un caso ideal enfrentado al marcador, la detección de segmentos de línea falla simultaneamente en los segmentos verticales como en los horitonzales. De otra forma se tendría una dirección que limita mas que la otra desaprovechando resolución.

La disposición espacial de los *conjuntos de cuadriláteros* esta en primer lugar limitada a un plano y en segundo lugar es tal que se puede definir ejes de coordenadas ortogonales mediante los centros como se muestra en la figura 2.3.

Por otro lado se tiene otro juego de parámetros **dinámicos** que concluyen con el diseño del marcador. Estos parámetros conservan la estructura intrínseca del marcador permitiendo versatilidad en la aplicación y sin la necesidad de modificación alguna de los algoritmos desarrollados. Estos son:

- d_{ij}: distancia entre los *centros* $QlSet[j]$ con $QlSet[i]$.
- l: lado del *cuadrilátero* mas pequeño ($Ql[0]$) de los $QlSet$.

En este caso se debe cumplir siempre la condición impuesta previamente en donde $d_{01} \gg d_{02}$. De otra forma se deberán realizar ciertas hipótesis no genéricas o se deberá aumentar ligeramente la complejidad del algoritmo para la identificación del marcador.

2.7.2.2. Diseños utilizados

- **Test**: Durante el desarrollo de los algoritmos de detección e identificación de los *vértices* del *marcador QR* se trabajó con determinados parámetros de diseño de dimensiones apropiadas para posibilitar el traslado y las pruebas domésticas.
- **Da Vinci**
- **Artigas**
- **Mapa**

2.7.3. Detección

La etapa de detección del marcador se puede separar en tres grandes bloques; la detección de segmentos de línea, el filtrado de segmentos y la determinación de correspondencias (figura **??**). En esta sección se muestran algunos resultados para la detección de segmentos de línea por LSD y se desarrolla en profundidad los algoritmos desarrollados durante el proyecto para el filtrado de segmentos y determinación de correspondencias.

2.7.3.1. Detección de segmentos de línea

La detección de segmentos de línea se realiza mediante el uso del algoritmo *LSD* el cual se detalla en el capítulo **??**. En forma resumida, dicho algoritmo toma como entrada una imagen en escala de grises de tamaño $W \times H$ y devuelve una lista de segmentos en forma de pares de puntos de origen y destino.

2.7.3.2. Filtrado de segmentos

El filtrado de segmentos consiste en la búsqueda de conjuntos de cuatro segmentos conexos en la lísta de segmentos de línea detectados por *LSD*. Los conjuntos de segmentos conexos encontrados se devuelven en una lista similar a la de *LSD*. A continuación se realiza una breve descripción del algoritmo de filtrado de segmentos implementado.

Se parte de una lista de m segmentos de línea,

$$\mathbf{L} = \begin{pmatrix} \mathbf{s}_0 & \mathbf{s}_1 & \dots & \mathbf{s}_{m-1} \end{pmatrix}^t \tag{2.2}$$

y se recorre en i en busca de segmentos vecinos. La estrategia utilizada consiste en buscar, para el i-ésimo segmento $\mathbf{s}_i$, dos segmentos vecinos, en primero lugar $\mathbf{s}_j$ y en segundo lugar $\mathbf{s}_k$, de forma que se forme una "U" como se muestra en la figura **??**. La tercer etapa de búsqueda consiste en completar ese conjunto con un cuarto segmento $\mathbf{s}_l$ que cierre la "U".

Dos segmentos son vecinos si se cumple que la distancia euclidiana entre puntos, d_{ij}, es menor a un cierto umbral para alguna de las combinaciones $\mathbf{p}_i \leftrightarrow \mathbf{p}_j$, $\mathbf{q}_i \leftrightarrow \mathbf{q}_j$, $\mathbf{p}_i \leftrightarrow \mathbf{q}_j$ o $\mathbf{q}_i \leftrightarrow \mathbf{p}_j$. En la primera etapa de la búsqueda se testean todas las posibilidades mientras que en la segunda etapa se testean solo los puntos del segmeneto que no fueron utilizados. Por ejemplo, si se encontró la correspondencia $\mathbf{p}_i \leftrightarrow \mathbf{p}_j$ se busca el k-ésimo segmento $\mathbf{s}_k$ que cumple que la distancia euclidiana d_{ij} es menor a cierto umbral para alguna de las combinaciones $\mathbf{q}_i \leftrightarrow \mathbf{p}_k$ y $\mathbf{q}_i \leftrightarrow \mathbf{q}_k$. En la tercer etapa la chequeo se realiza de forma mas aún mas restingida.

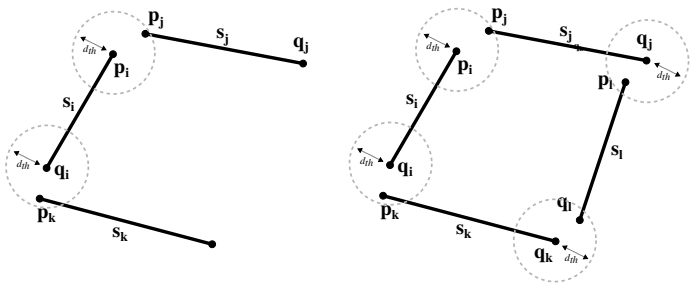

Figura 2.4: Conjunto de cuadriláteros conexos. A la izquiera el primer paso del filtrado en donde se busca una "U", a la derecha el último paso en donde se cierra la "U".

Una vez encontrado el conjunto de cuatro segmentos conexos se marcan estos segmentos como utilizados, se guardan en una lista de salida y se continúa con el segmento $i+1$ hasta recorrer los m segmentos de la lista de entrada. De esta forma se obtiene una lista de salida $\mathbf{S}$ de n segmentos en donde n es por construcción múltiplo de cuatro.

El algoritmo descripto es simple y provee resultados aceptables en general pero es propenso a tanto a detectar *falsos positivos* como al *sobre-filtrado* algunos conjuntos.

La detección de *falsos positivos* se puede atribuir prinicpalmente a la condición de vecindad utilizada en donde un caso como el que se muestra en la figura **??** de un conjunto de segmentos paralelos cercanos y de tamaño similar "sobrevive" al filtrado de segmentos. De forma de evitar estos falsos positivos, se podría considerar implementar un condición de vecindad que tome en cuenta la intersección de los segmentos y la distancia de la intersección a los puntos del segmento. Como se explicará en le sección **??**, debido a que el algoritmo de determinación de correspondencias realiza la intersección de estos segmentos se puede en ese momento filtrar estos casos.

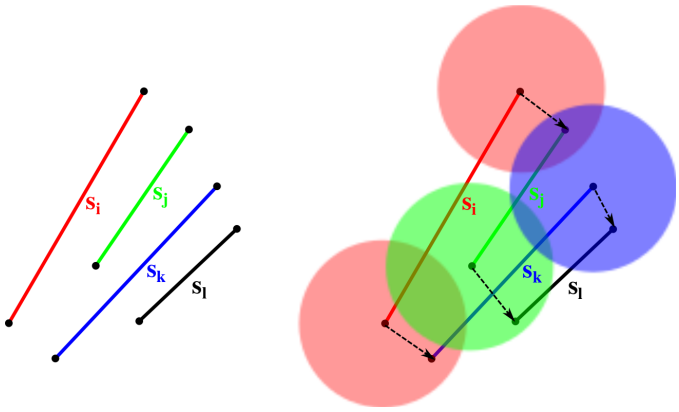

Figura 2.5: Posible configuración de segmentos paralelos que "sobreviven" al filtrado. A la izquiera el grupo de segmentos, a la derecha se muestra como se desarrolla el filtrado de $\mathbf{s}_i$.

El *sobre-filtrado* de segmentos tiende a ocurrir cuando no se cumple la condición de distancia entre segmentos vecinos cuando visualmente si lo son. Se debe principalmente a que se utiliza para el filtrado un valor de d_{th} fijo que resultó en buenos resultados para la aplicación pero en ciertas circunstancias produce estas fallas. Esta medida de distancia se podría tomar relativa a el largo del los segmentos a testear de forma de generalizar el valor pero se debería realizar analizar un poco mas en detalle la posibilidad de implementación para que resulte en buenos resultado y no introduzca otra clase de errores.

2.7.3.3. Determinación de correspondencias

Se detalla a continuación el algoritmo de determinación de correspondencias a partir de grupos de cuatro segmentos de línea conexos. Para ese algoritmo se hace uso de los elementos estructurales del marcado (sec.: **??**), de forma de desarrollar un algoritmo modular, escalable y simple.

Se toma como entrada la lista de segmentos filtrados

$$\mathbf{S} = \begin{pmatrix} \mathbf{s}_0 & \mathbf{s}_1 & \dots & \mathbf{s}_i & \mathbf{s}_{i+1} & \mathbf{s}_{i+2} & \mathbf{s}_{i+3} & \dots & \mathbf{s}_{n-1} \end{pmatrix}^t \tag{2.3}$$

en donde cada segmento se compone de un punto inicial $\mathbf{p}_i$ y un punto final $\mathbf{q}_i$, $\mathbf{s}_i = (\mathbf{p}_i, \mathbf{q}_i)$, con n es múltiplo de cuatro. Si i también lo es, entonces el sub-conjunto, $\mathbf{S}_i = \begin{pmatrix} \mathbf{s}_i & \mathbf{s}_{i+1} & \mathbf{s}_{i+2} & \mathbf{s}_{i+3} \end{pmatrix}^t$, corresponde a un conjunto de cuatro segmentos del línea conexos.

Para cada sub-conjunto $\mathbf{S}_i$ se intersectan entre sí los segmentos obteniendo una lista de cuatro vértices, $\mathbf{V}_i = \begin{pmatrix} \mathbf{v}_i & \mathbf{v}_{i+1} & \mathbf{v}_{i+2} & \mathbf{v}_{i+3} \end{pmatrix}^t$. Si $\mathbf{r}_i$ es la recta que pasa por los puntos $\mathbf{p}_i$ y $\mathbf{q}_i$ del segmento $\mathbf{s}_i$, la lista de vértices se obtiene como sigue,

$$\begin{aligned} \mathbf{v}_i &= \mathbf{r}_i \cap \mathbf{r}_{i+1} \\ \mathbf{v}_{i+1} &= \mathbf{r}_i \cap \mathbf{r}_{i+2} \\ \mathbf{v}_{i+2} &= \mathbf{r}_{i+3} \cap \mathbf{r}_{i+2} \\ \mathbf{v}_{i+3} &= \mathbf{r}_{i+3} \cap \mathbf{r}_{i+1} \end{aligned}$$

resultando en dos posibles configuraciones de vértices. Las dos configuraciones se muestran en la figura 2.4 en donde una de ellas tiene sentido horario y la otra antihorario partiendo de v_i.

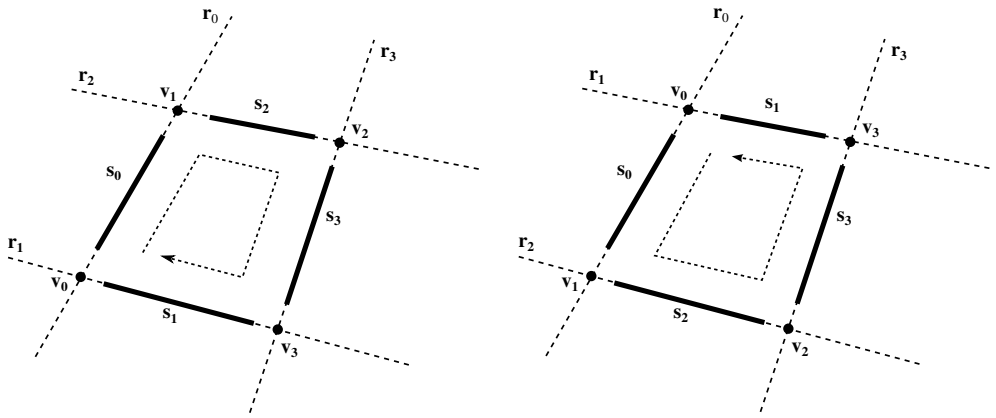

Figura 2.6: Posibles configuraciones de vértices posterior a la intersección de conjuntos de segmentos pertenecientes a un cuadrilátero.

Posterior a la intersección se realiza un chequeo sobre el valor de las coordenadas de lo vértices. Si alguno de ellos se encuentra fuera de los límites de la imagen, el conjunto de cuatro segmentos es marcado como inválido. Este chequeo resulta en el filtado de "falsos cuadriláteros" como por ejemplo un grupo de segmentos paralelos cercanos.

Para cada uno de los conjuntos de vértices se construye con ellos un elemento *cuadrilátero* que se almacena en una lista de cuadriláteros

$$QlList = \begin{pmatrix} Ql[0] & Ql[1] & \dots & Ql[i] & \dots & Ql[\frac{n}{4}] \end{pmatrix}^t$$

A partir de esa lista de cuadriláteros, se buscan grupos de tres cuadriláteros QlSet que "compartan" un mismo centro. Para esto se recorre ordenadamente la lista en i buscando para cada

cuadrilátero dos cuadriláteros j y k que cumplan que la distancia entre sus centros y el del i-ésimo cuadrilátero sea menor a cierto umbral d_{th},

$$d_{ij} = ||\mathbf{c}_i - \mathbf{c}_j|| < d_{th}, \quad d_{ik} = ||\mathbf{c}_i - \mathbf{c}_k|| < d_{th}. \tag{2.4}$$

Estos cuadriláteros se marcan en la lista como utilizados con ellos se forma el l-ésimo *QlSet* ordenándolos según su perímetro, de menor a mayor como

$$QlSet[l] = \begin{pmatrix} Ql[0] & Ql[1] & Ql[2] \end{pmatrix}$$

con $l = (0,1,2)$. Esta busqueda se realiza hasta encontrar un total de tres *QlSet* completos de forma de obtener un marcador completo, esto es, detectando todos los cuadriláteros que lo componen.

Una vez obtenida la lista de tres *QlSet*,

$$QlSetList = \begin{pmatrix} QlSet[0] & QlSet[1] & QlSet[2] \end{pmatrix}$$

ésta se ordena de forma que su dispoción espacial se corresponda con la del *marcador QR*. Para esto se calculan las distancias entre los centros de cada *QlSet* y se toma el índice i como el índice que produce el vector de menor distancia, $\mathbf{u}_i = \mathbf{c}_{i+1} - \mathbf{c}_i$. En en este punto que es importante que la condición de distancia entre los centros de los *QlSet* se cumpla, $d_{10} \gg d_{20}$, para una simple identificación. Bajo una transformación proyectiva del marcador, es posible que esta relación se modifique e incluso que deje de valer pero imponiendo la condición de mucho mayor nos aseguramos que el algoritmo funciona correctamente para condiciones razonables. Esto es, para proyecciones o poses que se encuentran dentro de las hipótesis uso de la aplicación.

Una vez seleccionado el vector $\mathbf{u}_i$, se tienen obtiene el juego de vectores $(\mathbf{u}_i, \mathbf{u}_{i+1}, \mathbf{u}_{i+2})$ como se muestra en la figura 2.5.

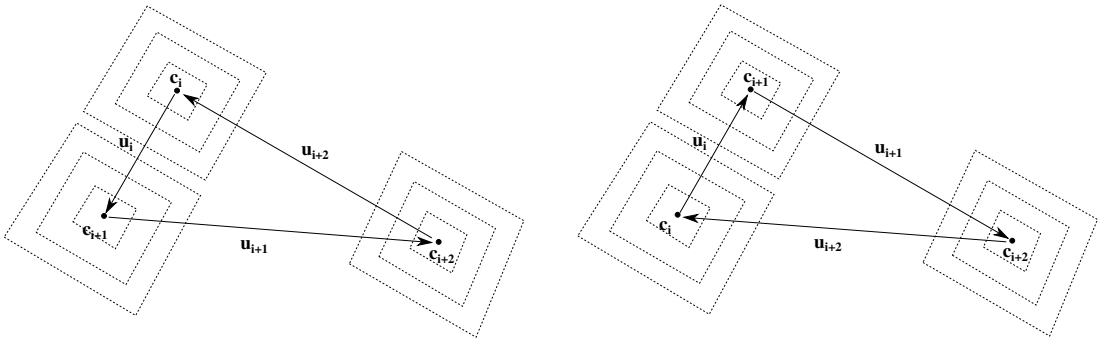

Figura 2.7: Posibles configuraciones de centros resultan en la orientación de los vectores $\mathbf{u}_{i+k}$.

Existen solo dos posibles configuraciones para estos vectores por lo que se utiliza este conocimiento para ordenar los *QlSet* de la lista realizando el producto vectorial, aumentando la dimensión de los vectores $\hat{\mathbf{u}}_i$ y $\hat{\mathbf{u}_{i+1}}$ con coordenada $z = 0$,

$$\mathbf{b} = \hat{\mathbf{u}}_i \times \hat{\mathbf{u}_{i+1}}.$$

Si el vector $\mathbf{b}$ tiene valor en la coordenada z positivo se ordena como,

$$\begin{aligned} QlSet[0] &\longleftarrow QlSet[i] \\ QlSet[1] &\longleftarrow QlSet[i+2] \\ QlSet[2] &\longleftarrow QlSet[i+1] \end{aligned}$$

o de lo contratio se ordena como,

$$\begin{aligned} QlSet[0] &\longleftarrow QlSet[i+1] \\ QlSet[1] &\longleftarrow QlSet[i+2] \\ QlSet[2] &\longleftarrow QlSet[i] \end{aligned}$$

Por ultimo se construye un *marcador QR* que contiene la lista de tres *QlSet* ordenados según lo indicado permitiendo la definición de un centro de coordenadas como el centro $\mathbf{c}_0$ del $QlSet[0]$ y ejes de coordenadas definidos en **??**. Los ejes de este sistema de coordenadas permiten, para cada *Ql* de cada *QlSet*, proyectar los vértices sobre sus ejes y según su signo ordenarlos como se muestra en la figura **??**. De esta forma, recorriendo ordenadamente los elementos del marcador, se ordenan

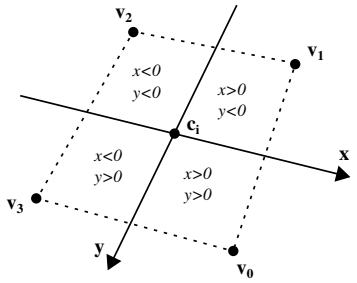

Figura 2.8: Posibles configuraciones de centros resultan en la orientación de los vectores $\mathbf{u}_{i+k}$.

los vértices detectados obteniendo una lista de vértices que se corresponde con la lista de vértices del marcador en coordenadas del mundo.

Se determinan las correspondencias $\mathbf{M}_i \leftrightarrow \mathbf{m}_i$ necesarias para la estimación de pose.

2.7.3.4. Robustificando la detección

El algorítmo descripto al momento requiere que dentro de la lista de segmentos filtrados se encuentren todos los segmentos que componen el marcador pero este requerimiento representa un problema importante en cuanto a el desempeño del algoritmo. En caso de que esto no se cumpla no es posible proporcionar las correspondencias necesarias para la estimación de pose y no se tendrá una pose válida para ese cuadro o *frame* para la aplicación. En aplicaciones en tiempo real en donde el procesamiento de la imagen es quien consume el mayor tiempo esto prejudica notablemente la fluidez resultando en que el sistema sea incomodo e incluso inutilizable. Es por esto que en esta sección se desarrolla la extensión del algoritmo de determinación de correspondencias para una cantidad menor de segmentos detectados y filtrados que resulta en una mejor sustancial en la cantidad de *frames* en los cuales es posible determinar correspondencias y obtener así una pose válida.

Se busca una determinación de correspondencias mas robusta pero manteniendo las esencia del algoritmo desarrollado. Por esto se tienen dos aspectos a tomar en cuenta; la detección de *QlSet*'s se realiza basada en la búsqueda de cuadriláteros concéntricos por lo que se debe contar con un mínimo de dos cuadriláteros por *QlSet* para permitir la diferenciación de entre un conjunto de segmentos filtrados debido a que pertenecen al marcador y a otro conjunto que no pertenece pero si cumple con las condiciones, por ejemplo podría ser el marco de una obra o cualquier elemento en la escena que forme un cuadrilátero. Esto fija un límite de no menos de 24 segmentos necesarios para el funcionamiento. El otro aspecto a tomar en cuenta se refiere a la forma en que se ordenan los *Ql*'s dentro de cada *QlSet*. Como ya se explicó el orden se basa en la medida del perímetro de los *Ql*'s ordenandolos de menor a mayor por lo que será necesario contar con, al menos, un *QlSet* completo

de forma de tener una referencia a la hora de identificar los *QlSet*'s incompletos hallados. Por lo tanto la extensión del algoritmo permite una correcta identificación de los vértices del marcador con un número mayor o igual a 28 segmentos.

La implementación de esta extensión del algoritmo se realizó manteniendo la estructura básica descrita anteriormente y se detalla aquí solamente los agregados realizados.

Al realizar la busqueda de conjuntos de cuadriláteros concéntricos se buscan en primer lugar los *QlSet*'s completos y luego en caso de que estos no lleguen a ser tres, se intenta completar buscando *QlSet*'s incompletos o sea conjuntos de dos cuadriláteros que comparten un mismo centro. Estos se agrupan en una lista de la misma forma en que se describió anteriormente pero dejando el tercer cuadrilátero, $Ql[2]$, marcado como inválido.

Una vez completada la lista de tres *QlSet* con por lo menos uno de ellos detectado completo se realiza ordenan en primer lugar los *QlSet* completos y de ellos se extrae una lista de perímetros promedio. Esta lista de perímetros promedio se utiliza para el ordenamiento de los *QlSet* incompletos comparando con los perímetros de los $Ql[0]$ y $Ql[1]$ de cada *QlSet*. El $Ql[2]$ previamente marcado como inválido se posiciona por descarte en la posición que corresponda.

Al momento de proporcionar la lista de vértices ordenados $\mathbf{m}_i$ y correspondientes con los del modelo $\mathbf{M}_i$, se introducen valores inválidos para los *Ql*'s marcados como inválidos. Por último se realiza un recorte de las dos listas de puntos en base a estos valores inválidos, se recorre la lista de puntos en la imagen $\mathbf{m}_i$ y se extraen de la lista de puntos en la imagen y de los puntos del modelo los puntos inválidos obteniendo un juego de al menos 28 correspondencias $\mathbf{m}'_i \leftrightarrow \mathbf{M}'_i$ para el algoritmo de estimación de pose.

2.7.3.5. Resultados

Oh sí!

CAPÍTULO 3

LSD: "Line Segment Detection"

3.1. 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.

3.2. *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.

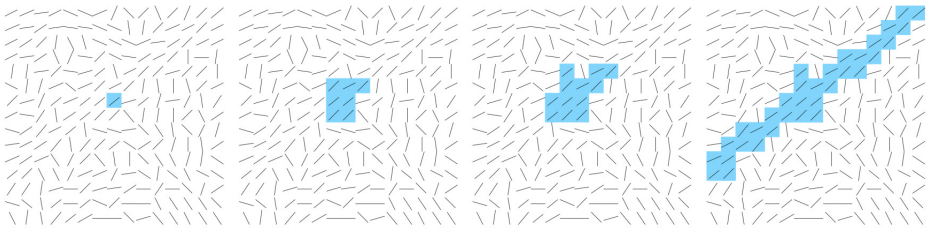

Figura 3.1: Proceso de crecimiento de una región. El ángulo asociado cada píxel de la imagen está representado por los pequeños segmentos y los píxeles coloreados representan la formación de la región. Fuente [7].

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 3.2. Este potencial problema es heredado del algoritmo de Burns, Hanson y Riseman.

Figura 3.2: 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.

3.3. 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 3.3: 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].

3.4. 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.

3.5. 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.

3.6. 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.

3.6.1. 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 3.4.

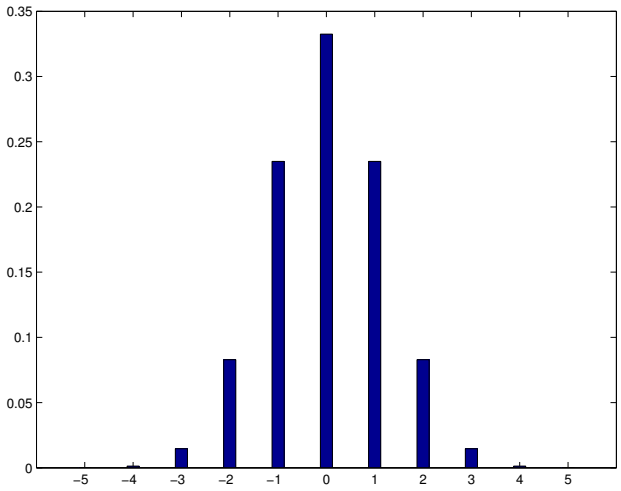

Figura 3.4: Núcleo Gaussiano utilizado por LSD. $\sigma = 1,2$.

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 3.5 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

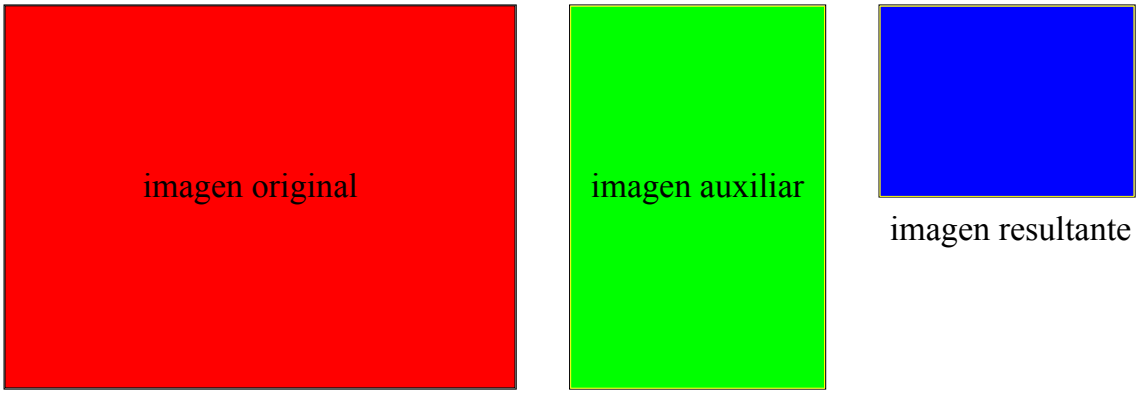

Figura 3.5: Relación entre las imágenes en consideradas en el filtro Gaussiano. Escala: $0,5$.

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 3.4); 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 3.6.

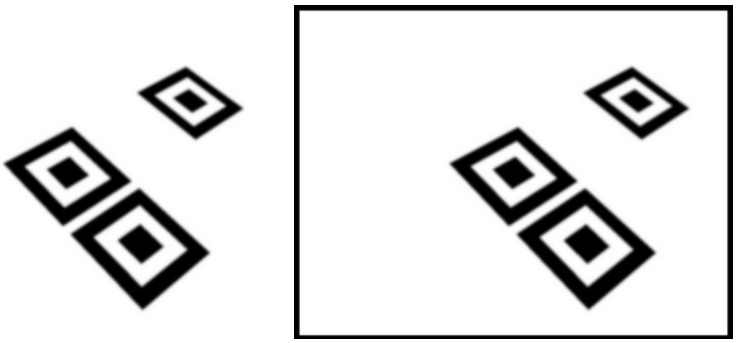

Figura 3.6: Imagen artificial del marcador trasladado y rotado, filtrada con el filtro Gaussiano. Izq.: Filtro Original. Der.: Filtro sin las condiciones de borde.

3.6.2. *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 3.6), que de no ser así se detectarían.

3.6.3. 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.

3.6.4. 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.

3.6.5. Resultados

3.6.5.1. Filtro Gaussiano

Figura 3.7: 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 3.7; 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 3.1 y 3.2:

- **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 3.1: 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 3.2: Comparación entre los tiempos de ejecución del filtro Gaussiano optimizado y el original. Ambos comprecisión simple.

3.6.5.2. *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 3.8: Imagen *zebras.png*.

ya que el algoritmo original está implementado con éstos integrados. Las imágenes de prueba fueron la del marcador sintético (figura 3.7) y *zebras.png* mostrada en la figura 3.8. Los valores relevantes de los experimentos se muestran en las tablas 3.3, 3.4, 3.5 y 3.6.

- **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 3.3: Comparación entre los tiempos de ejecución del filtro Gaussiano más LSD optimizados y los originales, para la imagen 3.7. 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 3.4: Comparación entre los tiempos de ejecución del filtro Gaussiano más LSD optimizados y los originales, para la imagen 3.8. 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 3.5: Comparación entre los tiempos de ejecución del filtro Gaussiano más LSD optimizados y los originales, para la imagen 3.7. 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 3.6: Comparación entre los tiempos de ejecución del filtro Gaussiano más LSD optimizados y los originales, para la imagen 3.8. En todos los casos con comprecisión simple.

CAPÍTULO 4

Modelo de cámara y estimación de pose monocular

4.1. Introducción

Se le llama "estimación de pose" al proceso mediente el cual se calcula en qué punto del mundo y con qué orientación se encuentra determinado objeto respecto de un eje de coordenadas previamente definido al que se lo llama "ejes del mundo". Las aplicaciones de realidad aumentada requieren de un modelado preciso del entorno respecto de estos ejes, para poder ubicar correctamente los agregados virtuales dentro del modelo y luego dibujarlos de forma coherente en la imagen vista por el usuario. El objeto cuya estimación de pose resulta de mayor importancia es la cámara, ya que por ésta es por donde se mira la escena y es respecto de ésta que los objetos virtuales deben ubicarse de manera consistente. Una forma de estimar la pose de la cámara es mediante el uso de las imágenes capturadas por ella misma.
Asimismo, el concepto "monocular" hace referencia al uso de una sola cámara, ya que es posible trabajar con más de una.
Para poder obtener información relevante a partir de las imágenes tomadas por una cámara, resulta necesario contar con un modelo preciso de su arquitectura ya que no todas las cámaras son iguales. El modelo más comunmente utilizado es el denominado *pin-hole*. Para modelar completamente la arquitectura de la cámara se deben estimar ciertos "parámetros intrínsecos" a ésta, y eso se logra luego de realizados ciertos experimentos. A la estimación de estos parámetros se le denomina "calibración de la cámara".

4.2. Calibración de cámara: modelo pin-hole [1]

4.2.1. Fundamentos y definiciones

Este modelo consiste en un centro óptico C, en donde convergen todos los rayos de la proyección y un plano imagen en el cual la imagen es proyectada. Se define "distancia focal" (f) como la distancia entre el centro óptico C y el cruce del eje óptico por el plano imagen (punto P). Ver imagen 4.1.

Para modelar el proceso de proyección (proceso en el que se asocia al punto **M** del mundo, un punto **m** en la imagen), es necesario referirse a varias transformaciones y varios ejes de coordenadas.

- *Coordenadas del mundo*: son las coordenadas que describen la posición 3D del punto **M**. Se definen respecto de los *ejes del mundo* (X_m, Y_m, Z_m). La elección de los ejes del mundo es

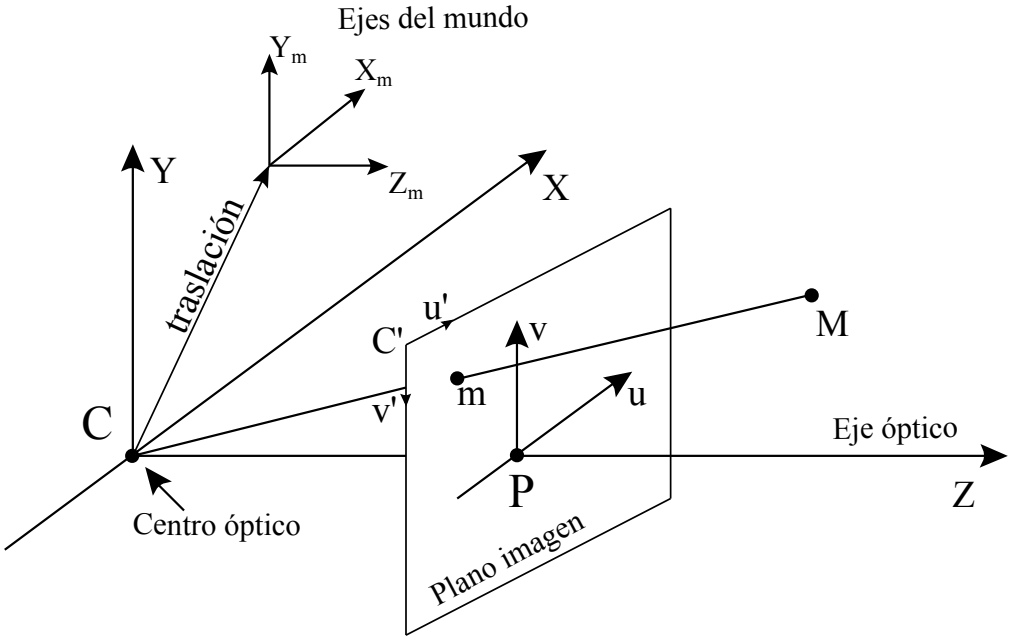

Figura 4.1: Modelo de cámara pin-hole.

arbitraria.

- *Coordenadas de la cámara*: son las coordenadas que describen la posición del punto **M** respecto de los ejes de la cámara (X,Y,Z).
- *Coordenadas de la imagen*: son las coordenadas que describen la posición del punto 2D, **m**, respecto del centro del plano imagen, P. Los ejes de este sistema de coordenadas son (u,v).
- *Coordenadas normalizadas de la imagen*: son las coordenadas que describen la posición del punto 2D, **m**, respecto del eje de coordenadas (u',v') situado en la esquina superior izquierda del plano imagen.

La transformación que lleva al punto **M**, expresado respecto de los ejes del mundo, al punto **m**, expresado respecto del sistema de coordenadas normalizadas de la imagen, se puede ver como la composición de dos transformaciones menores. La primera, es la que realiza la proyección que transforma a un punto definido respecto del sistema de coordenadas de la cámara (X,Y,Z) en otro punto sobre el plano imagen expresado respecto del sistema de coordenadas normalizadas de la imagen (u',v'). Véase que una vez calculada esta transformación, es una constante característica de cada cámara. Al conjunto de valores que definen esta transformación, se le llama "parámetros intrínsecos" de la cámara. La segunda, es la transformación que lleva de expresar a un punto respecto de los ejes del mundo (X_m,Y_m,Z_m), a los ejes de la cámara (X,Y,Z). Esta última transformación varía conforme se mueve la cámara (respecto de los ejes del mundo) y el conjunto de valores que la definen es denominado "parámetros extrínsecos" de la cámara. Del cálculo de estos parámetros es que se obtiene la estimación de la pose de la cámara.
De lo anterior se concluye rápidamente que si se le llama H a la matriz proyección total, tal que:

$$m = H.M,$$

entonces:

$$H = I.E$$

donde *I* corresponde a la matriz proyección asociada a los parámetros intrínsecos y *E* corresponde a la matriz asociada a los parámetros extrínsecos. Ambos juegos de parámetros acarrean información muy valiosa:

- **Parámetros extrínsecos:** pose de la cámara.
  - Traslación: ubicación del centro óptico de la cámara respecto de los ejes del mundo.
  - Rotación: rotación del sistema de coordenadas de la cámara (X,Y,Z), respecto de los ejes del mundo.
- **Parámetros intrínsecos:** parámetros propios de la cámara. Dependen de su geometría interna y de su óptica.
  - Punto principal (P = $[u'_P, v'_P]$): es el punto intersección entre el eje óptico y el plano imagen. Las coordenadas de este punto vienen dadas en píxeles y son expresadas respecto del sistema normalizado de la imagen.
  - Factores de conversión píxel-milímetros (d_u, d_v): indican el número de píxeles por milímetro que utiliza la cámara en las direcciones *u* y *v* respectivamente.
  - Distancia focal (f): distancia entre el centro óptico (**C**) y el punto principa (**P**). Su unidad es el milímetro.
  - Factor de proporción (s): indica la proporción entre las dimensiones horizontal y vertical de un píxel.

4.2.2. Matriz de proyección

En la sección anterior se vió que es posible hallar una "matriz de proyección" H que dependa tanto de los parámetros intrínsecos de la cámara como se sus parámetros extrínsecos:

$$m = H.M$$

donde **M** y **m** son los puntos ya definidos y vienen expresados en "coordenadas homogéneas". Por más información acerca de este tipo de coordenadas ver [8].
Para determinar la forma de la matriz de proyección se estudia cómo se relacionan las coordenadas de **M** con las coordenads de **m**; para hallar esta relación se debe analizar cada transformación, entre los sistemas de coordenadas mencionados con anterioridad, por separado.

- **Proyección 3D - 2D:** de las coordenadas homogéneas del punto **M** expresadas en el sistema de coordenadas de la cámara (X_0, Y_0, Z_0, T_0), a las coordenadas homogéneas del punto **m** expresadas en el sistema de coordenadas de la imagen (u_0, v_0, s_0):
  Se desprende de la imagen 4.1 y algo de geometría proyectiva la siguiente relación entre las coordenadas en cuestión y la distancia focal (f):

$$\frac{f}{Z_0} = \frac{u_0}{X_0} = \frac{v_0}{Y_0}$$

  A partir de la relación anterior:

$$\begin{pmatrix} u_0 \\ v_0 \end{pmatrix} = \frac{f}{Z_0} \begin{pmatrix} X_0 \\ Y_0 \end{pmatrix}$$

Expresado en forma matricial, en coordenadas homogéneas:

$$\begin{pmatrix} u_0 \\ v_0 \\ s_0 \end{pmatrix} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X_0 \\ Y_0 \\ Z_0 \\ 1 \end{pmatrix}$$

- **Transformación imagen - imagen:** de las coordenadas homogéneas del punto **m** expresadas respecto del sistema de coordenadas de la imagen (u_0, v_0, s_0), a las coordenadas homogéneas de él mismo pero expresadas respecto del sistema de coordenadas normalizadas de la imagen (u'_0, v'_0, s'_0):

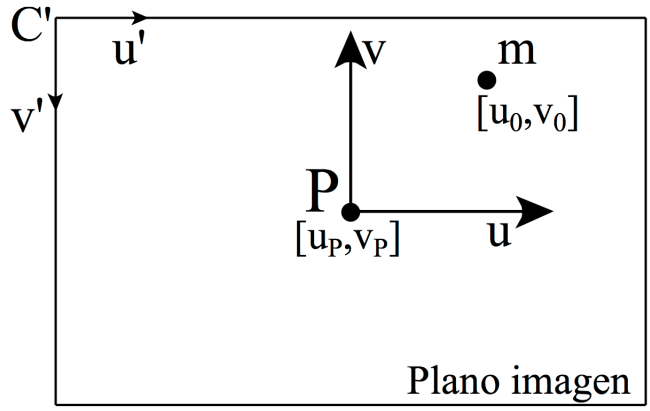

Figura 4.2: Relación entre el sistema de coordenadas de la imagen y el sistema de coordenandas normalizadas de la imagen.

Se les suma, a las coordenadas de **m** respecto del sistema de la imagen, la posición del punto P respecto del sistema normalizado de la imgen (u'_P, v'_P). Las coordenadas de **m** dejan de ser expresadas en milímetros para ser expresadas en píxeles. Aparecen los factores de conversión d_u y d_v:

$$u'_0 = d_u.u_0 + u'_P$$
$$v'_0 = d_v.v_0 + v'_P$$

Se obtiene entonces la siguiente relación matricial, en coordenadas homogéneas:

$$\begin{pmatrix} u'_0 \\ v'_0 \\ s'_0 \end{pmatrix} = \begin{pmatrix} d_u & 0 & u'_P \\ 0 & d_v & v'_P \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u_0 \\ v_0 \\ 1 \end{pmatrix}$$

- **Matriz de parámetros intrínsecos** (I)**:** de las coordenadas homogéneas del punto **M** expresadas en el sistema de coordenadas de la cámara (X_0, Y_0, Z_0, T_0), a las coordenadas homogéneas del punto **m** expresadas respecto del sistema de coordenadas normalizadas de la imagen (u'_0, v'_0, s'_0):
  Se obtiene combinando las dos últimas transformaciones. Nótese que como ya se aclaró, depende únicamente de parámetros propios de la construcción de la cámara:

$$I = \begin{pmatrix} d_u.f & 0 & u'_P & 0 \\ 0 & d_v.f & v'_P & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Nota: De forma genérica se puede agregar a la matriz de parámetros intrínsecos del modelo *pin-hole* un parámetro **s** llamado en inglés *skew parameter*, o "parámetro de proporción" en Español. Este parámetro toma valores distintos de cero muy rara vez, pues modela los casos en los que los ejes x e y de los píxeles de la cámara no son perpendiculares entre sí. En casos realistas, $s \neq 0$ cuando por ejemplo se toma una fotografía de una fotografía. La matriz de parámetros intrínsecos, tomando en cuenta este parámetro, tendrá la forma:

$$I = \begin{pmatrix} d_u.f & s & u'_P & 0 \\ 0 & d_v.f & v'_P & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

- **Matriz de parámetros extrínsecos** (E)**:** de las coordenadas homogéneas del punto **M** expresadas respecto del sistema de coordenadas del mundo $(X_{m0}, Y_{m0}, Z_{m0}, T_{m0})$, a las coordenadas homogéneas de él mismo pero expresadas respecto del sistema de coordenadas de la cámara (X_0, Y_0, Z_0, T_0):
  Se obtiene de estimar la pose de la cámara respecto de los ejes del mundo y es la combinación de, primero una rotación R, y luego una traslación T. Se obtiene entonces la siguiente representación matricial:

$$\begin{pmatrix} X_0 \\ Y_0 \\ Z_0 \\ T_0 \end{pmatrix} = \begin{pmatrix} R & T \\ 0 & 1 \end{pmatrix} \begin{pmatrix} X_{m0} \\ Y_{m0} \\ Z_{m0} \\ T_{m0} \end{pmatrix}$$

  donde la matriz de parámetros extrínsecos desarrollada toma la forma:

$$E = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- **Matriz de proyección** (H)**:** de las coordenadas homogéneas del punto **M** expresadas respecto del sistema de coordenadas del mundo $(X_{m0}, Y_{m0}, Z_{m0}, T_{m0})$, a las coordenadas homogéneas del punto **m** expresadas respecto del sistema de coordenadas normalizadas de la imagen (u'_0, v'_0, s'_0):
  Es la proyección total y se obtiene combinando las dos transformaciones anteriores:

$$\begin{pmatrix} u'_0 \\ v'_0 \\ s'_0 \end{pmatrix} = \begin{pmatrix} d_u.f & 0 & u'_P & 0 \\ 0 & d_v.f & v'_P & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} . \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} . \begin{pmatrix} X_{m0} \\ Y_{m0} \\ Z_{m0} \\ T_{m0} \end{pmatrix}$$

4.3. Distorsión introducida por las lentes

Hasta el momento se asumió que el modelo lineal presentado para la proyección de cualquier punto del mundo en el plano imagen de la cámara es lo suficientemente preciso en todos los casos. Sin embargo, en casos reales, y cuando las lentes de las cámaras no son del todo buenas, la distorsión introducida por estas se hace notar. Dado el punto **M** de coordenadas (X_0, Y_0, Z_0) respecto de los ejes de la cámara, se le llama distorsión a la diferencia entre su proyección ideal en el plano