v0.3

# CAPÍTULO 1

# Detección

## 1.1. Introuducción

En este Capítulo se trata con algunas de las herramientas y algoritmos de detección de características del tipo de primitivas geométricas como ser puntos, esquinas, bordes, líneas, segmentos de línea y arcos.

El objetivo principal es mostrar qué características se pueden detectar y con la utilización de qué algoritmos con el fin de su utilización en un algoritmo de estimación pose, tema que se desarrolla en los Capítulos **??** y, en forma más específica en el Capítulo **??**. El tipo de características elegidas para su detección en la aplicación final están fuertemente ligadas con el desempeño del algoritmo que realiza dicha detección. Existe un requerimiento de tiempo ya que la aplicación final se debe ejecutar en tiempo real. También se desea que la característica ofrezca la posibilidad de ser procesada posteriormente mediante algoritmos relativamente simples.

Se busca que las características detectadas, asociadas a objetos o elementos presentes en la escena, presenten estabilidad en su ubicación frente a cambios en la pose. En forma más específica, si se tiene una esquina a detectar en una escena, por ejemplo de una mesa, la esquina puede ser detectada frente a imágenes o "fotos" de la mesa desde distintas posiciones y que en todos los casos la esquina detectada corresponde con la esquina de la mesa en la foto. También se busca que las características sean robustas frente a cambios en las condiciones en que se adquiere la imagen. Estas condiciones podrían ser iluminación, presencia de ruido, oclusiones, etc.

El capítulo comienza con algunos detectores clásicos de bordes y esquinas como son Canny y Harris. Posteriormente se desarrollan algunos detectores de líneas, segmentos de línea y algunas estructuras más complejas, como el Método de la Transformada de Hough, LSD, EDLines y ORT. A lo largo de todo el capítulo se presentan resultados de estos algoritmos

## 1.2. Bordes y esquinas

Los detectores de bordes, en particular los detectores de bordes tipo escalón son una parte esencial de muchos sistemas de visión. El proceso de detección de bordes cumple la función de simplificar drásticamente la cantidad de información a procesar conservando aspectos estructurales utilizables para establecer los límites de cierto objeto en la imagen [**?**].

Un **borde** se puede definir intuitivamente como un cambio brusco sobre una zona en la intensidad de una imagen. De forma más formal puede ser asociado a la búsqueda de discontinuidades y naturalmente su implementación está relacionada con la derivada primera de una imagen. En una imagen una derivada primera puede ser aproximada de distintas formas por un operador de gra-

diente como lo son los operadores de Sobel, Prewitt o Roberts por ejemplo. Hay implementaciones de detección de bordes que toman en cuenta el *efecto rampa* y por lo tanto acuden a derivadas de segundo y tercer orden. El efecto rampa se muestra en la Figura 1.1 en donde se puede ver en lo que aparentemente ocurre una discontinuidad en la imagen si uno se acerca lo suficiente a la región, esta discontinuidad es en realidad continua.

Figura 1.1: Efecto rampa. A la izquierda una imagen que presenta una discontinuidad. A la derecha una acercamiento en la región marcada en rojo.

### 1.2.1. Detector de bordes de Canny

El detector de bordes de Canny fue desarrollado en 1986 por John F. Canny[**?**] en el artículo *"A Computational Approach to Edge Detection"* con el objetivo de descubrir el algoritmo de detección de bordes óptimo. En esta Sección se describe brevemente el algoritmo y se muestran algunos resultados de interés. El desarrollo está inspirado principalmente en el artículo del detector de bordes de Canny de Wikipedia[1].

Las características buscadas por Canny para el detector de borde óptimo son:

- **Buena detección**: El algoritmo debe marcar todos los bordes reales en la imagen como sea posible.
- **Buena localización**: Los bordes marcados deben estar lo más cercanos posibles a los bordes reales en la imagen.
- **Respuesta mínima**: Dado un borde en la imagen este no debe ser marcado más de una vez y de ser posible la presencia de ruido no debe crear falsos bordes.

En la Figura 1.2(b) se muestra un ejemplo del detector de bordes de Canny para la imagen de Lena original1.2(a).

El algoritmo consiste en cuatro etapas básicas las cuales se describen a continuación. A lo largo de estas etapas se ejemplifica mediante el uso del *demo on-line* de Canny en IPOL[41] para la imagen de Lena. Estos resultados intermedio se encuentan en la Figura 1.3.

- **Reducción de Ruido**: Debido a que el detector de bordes de Canny es sensible a la presencia de ruido en las imágenes se utiliza en una primera etapa un filtrado Gaussiano en donde se realiza la convolución de la imagen con un núcleo Gaussiano. El resultado da lugar a una imagen levemente borrosa como se puede observar en la Figura 1.3(a).
- **Gradiente de intensidad de la imagen**: De forma de detectar bordes en todas las direcciones se utilizan cuatro filtros para detectar bordes en direcciones horizontal, vertical, y las dos

[1] `http://en.wikipedia.org/wiki/Canny_edge_detector`

(a) Imagen de entrada: Lena (b) Imagen de salida: Canny

Figura 1.2: Detector de bordes de Canny para la imagen Lena con parámetros: $\sigma = 1{,}6$, $th_{lo} = 4{,}0$, $th_{hi} = 20{,}0$

direcciones diagonales. El operador de detección de bordes, por ejemplo Roberts, Prewitt o Sobel, devuelve un valor para la derivada primera en la dirección horizontal ($\mathbf{G}_x$) y otro para la derivada primera en la dirección vertical ($\mathbf{G}_y$). En base a estos valores se puede obtener módulo y dirección del gradiente sobre cada píxel (Figura 1.3(b)).

$$\mathbf{G} = \sqrt{\mathbf{G}_x^2 + \mathbf{G}_y^2} \tag{1.1}$$

$$\Theta = \arctan\left(\frac{\mathbf{G}_y}{\mathbf{G}_x}\right) \tag{1.2}$$

El ángulo de dirección del borde es redondeado a uno de los cuatro ángulos (0, 45, 90 o 135 grados).

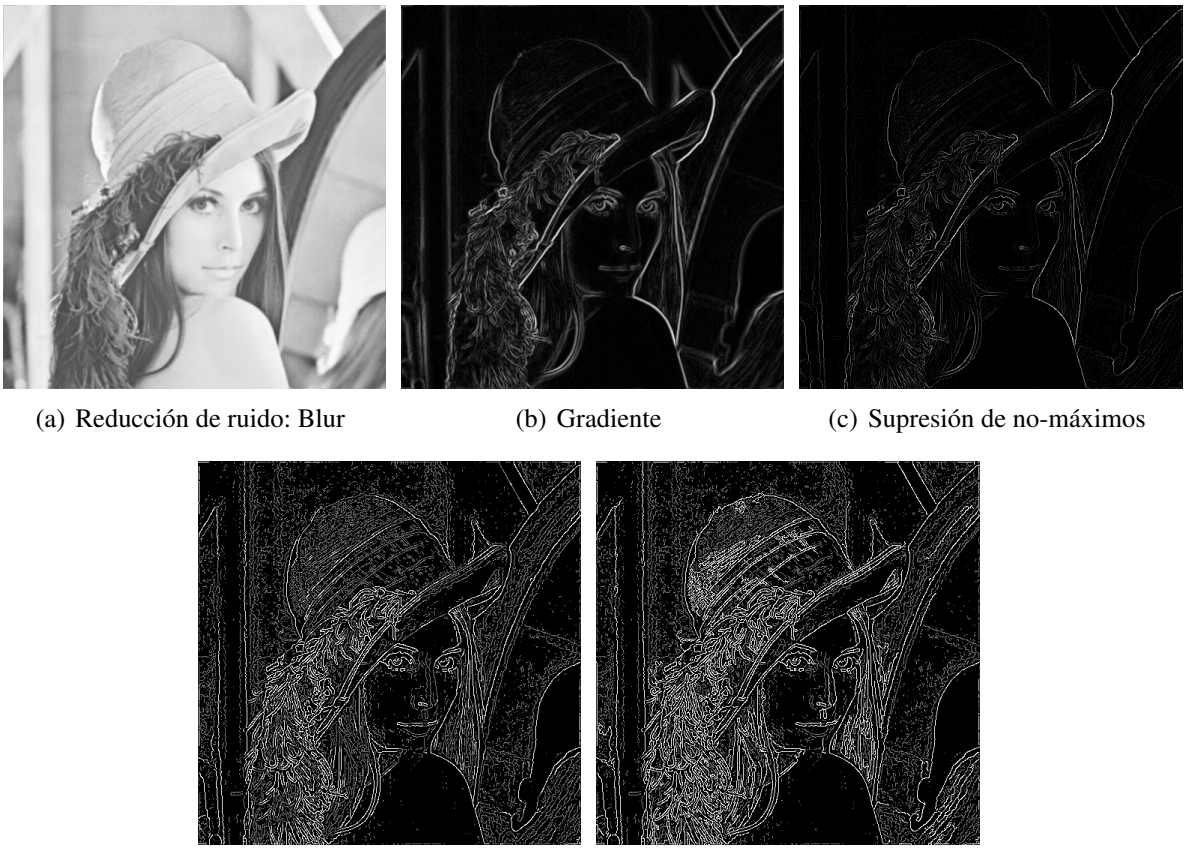
- **Supresión de no-máximos**: Dados los estimadores del gradiente de la imagen se realiza una búsqueda para determinar si el módulo del gradiente asume un rol de máximo local. De esta etapa se obtiene un conjunto puntos en forma de imagen binaria llamados "thin edges" como se puede ver en la Figura 1.3(c). Estos proporcionan información de dirección para realizar el umbralizado con histéresis.

  La búsqueda de máximos se realiza observando el ángulo del punto y el módulo de él y sus vecinos. Por ejemplo, si el ángulo del gradiente pertenece al conjunto de los de cero grados (el borde es vertical) se observa el módulo de sus vecinos horizontales. Si el módulo del gradiente del punto es mayor a los de estos entonces el punto asume un rol de máximo local. Los otros casos son análogos.

- **Umbrales con histéresis**: El umbralizado con histéresis requiere de dos umbrales, uno bajo y uno alto. El umbral alto asegura que se marcan bordes de los cuales se puede estar suficientemente seguros de que son borde (Figura 1.3(d)). Luego en base a los bordes correspondientes a umbrales altos se aplica el umbral bajo recorriendo los caminos de dirección obtenidos previamente. Una vez finalizado el proceso se tiene una imagen binaria en donde cada píxel es marcado como borde o no (Figura 1.3(e)).

El algoritmo consiste en una serie de parámetros que ya fueron mencionados pero que vale la pena resaltar. En primer lugar el parámetro $\sigma$, la desviación estandar, del filtro Gaussiano utilizado

en la etapa de reducción de ruido. Este parámetro afecta notablemente el desempeño del algoritmo y debe ser ajustado según la aplicación ya que aporta un compromiso entre el detalle para bordes más fino y la cantidad de falsos bordes producto del ruido. Por otro lado en la etapa de umbralizado con histéresis se tienen dos parámetros más, los valores de los dos umbrales; alto y bajo. En estos dos parámetros residen los típicos problemas de los umbrales y no hay un enfoque genérico que lo solucione, si el umbral alto es muy alto se puede perder información importante y por otro lado si el umbral bajo es muy bajo se producirán más falsos bordes debido a ruido.



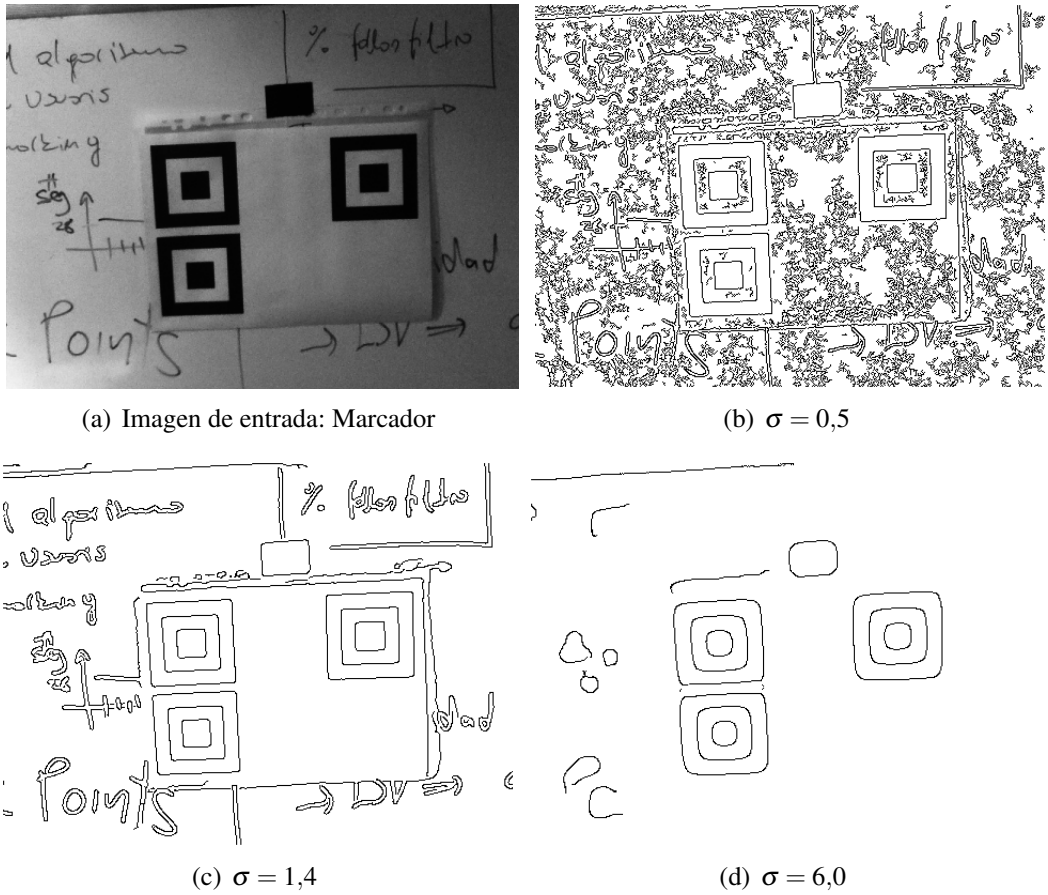
(a) Reducción de ruido: Blur (b) Gradiente (c) Supresión de no-máximos

(d) Umbralizado con histéresis: umbral alto (e) Umbralizado con histéresis: umbral bajo

Figura 1.3: Resultados intermedios del detector de bordes de Canny para la imagen Lena con parámetros: $\sigma = 1{,}6$, $th_{lo} = 4{,}0$, $th_{hi} = 20{,}0$

#### 1.2.1.1. Ejemplos de interés

En la Figura 1.4 se muestran distintos resultados de Canny para distinto valor de $\sigma$, utilizando como entrada una imagen conteniendo el marcador utilizado en el proyecto. Se puede ver que con $\sigma = 0{,}5$ la presencia de ruido afecta notablemente la detección produciendo una gran cantidad de falsos bordes. Para $\sigma = 1{,}4$ la detección es buena eliminando prácticamente todos los falsos bordes producto del ruido y conservando la información de borde real. Por otro lado para $\sigma = 6{,}0$ el *bluring* resulta en una detección de bordes muy reducida aunque preservando la información de bordes más significativos como los bordes del marcador los cuales son el interés principal en la aplicación. De todas formas se produce un efecto no deseado de "redondeo" de los bordes asociados al uso de un núcleo Gaussiano circular.



(a) Imagen de entrada: Marcador (b) $\sigma = 0{,}5$

(c) $\sigma = 1{,}4$ (d) $\sigma = 6{,}0$

Figura 1.4: Resultados del detector de bordes de Canny variando el parámetro de reducción de ruido $\sigma$ para una imagen de entrada conteniendo un marcador. $th_{lo} = 0{,}2$, $th_{hi} = 0{,}8$ (umbrales normalizados

### 1.2.2. Detector de bordes y esquinas de Harris

El detector de bordes y esquinas de **Harris** es un método desarrollado por Chris Harris et al. en 1988 en el artículo *"A combined corner and edge detector"*[21].

Se considera una ventana en el píxel $(x,y)$ en donde se define la suma cuadrada ponderada de los elementos de la ventana como

$$E_{x,y} = \sum_{u,v} w_{u,v}[I_{x+u,y+v} - I(u,v)]^2 = \sum_{u,v} w_{u,v}[xI_x + yI_y + O(x^2,y^2)]^2$$

en donde $w_{u,v}$ es un núcleo gaussiano para suavizar la respuesta y los gradientes pueden ser aproximados por

$$I_x = I * (-1,0,1) \qquad I_y = I * (-1,0,1)^T.$$

Para pequeños variaciones, se puede tomar la aproximación

$$E(x,y) = Ax^2 + 2Cxy + By^2$$

en donde $A = I_x^2 * w$, $B = I_y^2 * w$ y $C = I_x I_y * w$. De forma de tomar en cuenta la variación de $E$ junto con la dirección del cambio se reformula la ecuación anterior como

$$E(x,y) = \begin{pmatrix} x & y \end{pmatrix} \begin{pmatrix} A & C \\ C & B \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x & y \end{pmatrix} M \begin{pmatrix} x \\ y \end{pmatrix}$$

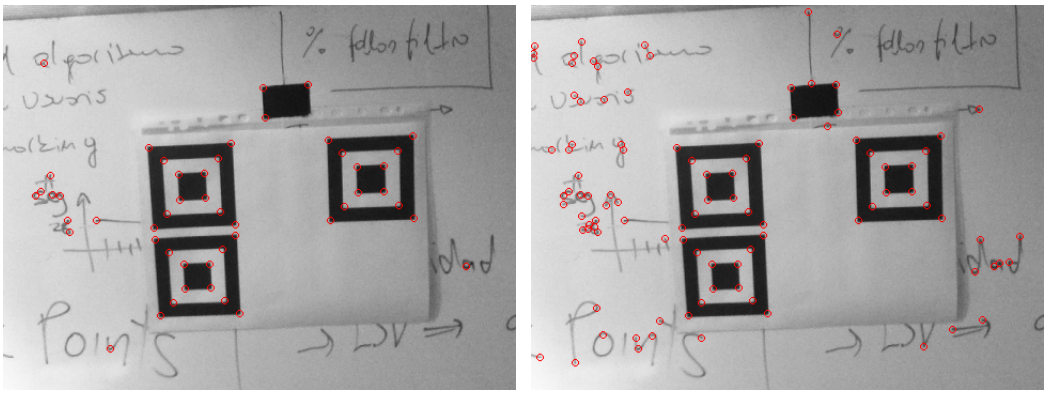
Dado que $E$ está fuertemente relacionado con la función de autocorrelación, si $\alpha$ y $\beta$ son los valores propios de la matriz $M$, serán proporcionales a la curvatura de la función de autocorrelación formando un descriptor invariante a rotaciones de $M$. La respuesta a esquinas propuesta para el detector de harris es computacionalmente más eficiente y no es necesario calcular valores propios de la matriz $M$. Esta dada por

$$R = det(M) - ktr^2(M)$$

en donde $k$ es una constante de sensibilidad. Las esquinas están representadas por los valores de $R$ positivos mientras que los bordes corresponden a los $R$ negativos y con $R$ cercano a cero tendremos zonas planas.

#### 1.2.2.1. Ejemplos de interés

En la Figura 1.6 se muestran los resultados para el detector de Harris implementado por Matlab por la función `corner`. La misma tiene un parámetro modificable asociado a la máxima cantidad de esquinas que se desean tener. Se puede ver que los resultados son buenos para ambos casos y



(a) Harris para $n = 50$ (b) Harris para $n = 100$

Figura 1.5: Resultados Harris para la imagen del marcaodor. El parámetro $n$ es un máximo en la cantidad de esquinas detectadas.

particularmente buenos si se desean detectar las esquinas del marcador.

#### 1.2.2.2. Aplicabilidad

Se pudo ver que el detector de Harris resultó en una detección razonable aunque su funcionamiento no es tan bueno bajo otras condiciones menos controladas ya que puede ser inestable y no muy robusto **¿fruta?**.

Una desventaja importante que tiene la detección de características tipo esquinas es que si se desea saber qué esquinas se unen entre sí mediante los lados del marcador sin ambiguedad se debe obtener más información que permita conectarlas. Por lo tanto un buen complemento puede ser el gradiente o una detección de bordes. La detección de líneas o segmentos ya contiene esta información y por lo tanto parece más apropiada.

## 1.3. Líneas y segmentos de línea

Las líneas o segmentos de línea son elementos que contienen una estructura más definida en comparación con los bordes. Esta estructura definida produce a su vez que la detección de este tipo

de elementos sea más estable y menos sensible al ruido que la detección de bordes. Como se verá en a lo largo de la sección, los detectores de líneas que se presentan tienen en común que utilizan al inicio de la detección las herramientas de gradientes y operadores gradientes o directamente la detección de bordes en sí misma para realizar el objetivo de detección de líneas.

En esta sección de explican brevemente algunos detectores de líneas y segmentos de línea. En primer lugar se menciona un detector de líneas clásico y popular como es el detector de Hough y se explica su funcionamiento así como sus principales desventajas que hacen que este método sea descartado para su desarrollo en el proyecto. En segundo lugar se menciona y se hace referencia al capítulo en donde será tratado el detector de segmentos de línea, uno de los pilares principales de este proyecto, el LSD. También se muestran otros detectores que fueron tomados en consideración pero por diferentes razones (explicadas en su secciones correspondientes) no fueron utilizados como son el ORT y EDLines.

### 1.3.1. Detector de líneas de Hough

El Método de la Transformada de Hough forma parte de una técnica de extracción de características con fuerte aplicación a detección de curvas y líneas.

Esta técnica consiste en tres pasos básicos. En primer lugar la imagen en niveles de gris es procesada por un detector de bordes devolviendo una máscara binaria en donde los puntos borde son marcados. En segundo lugar la Transformada de Hough es aplicada a la máscara de forma de detectar candidatos a líneas en forma de máximos locales. Esta transformada consiste en una representación paramétrica de formas geométricas. Para detección de líneas se utiliza la representación

$$r = xcos(\theta) + ysin(\theta) \tag{1.3}$$

en donde $r$ representa la distancia entre la línea y el origen y $\theta$ es el ángulo del vector perpendicular a la recta desde el origen. Es posible asociar una recta al par de parámetros $(r, \theta)$. El plano $(r, \theta)$ es es plano de Hough. Se determina en este plano para cada punto de la imagen una única sinusoidal en donde mediante su superposición se tendrá, para una serie de puntos alineados en la imagen, un punto de cruce de estas sinusoides dado por los parámetros $(r, \theta)$

Por lo tanto buscando máximos locales en el plano de Hough se tienen definidas líneas en la imagen.

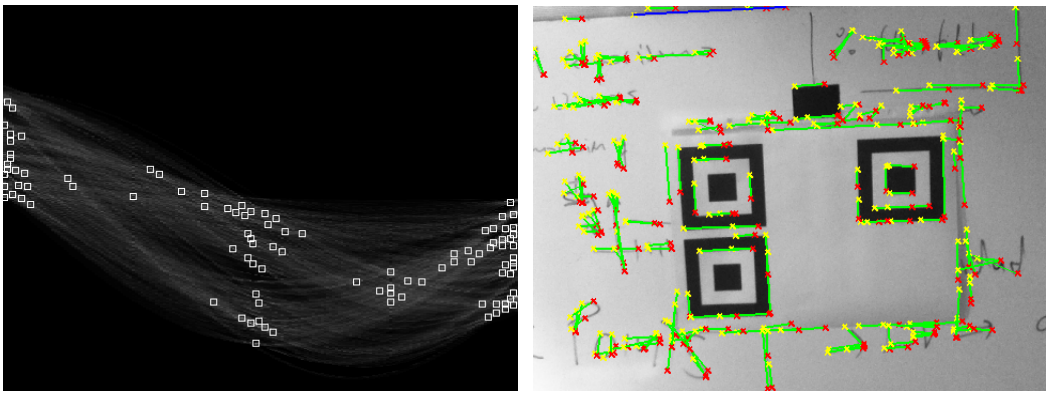
Finalmente los segmentos son extraídos de las líneas detectadas utilizando umbrales sobre dos parámetros geométricos: el largo mínimo de un segmento y la distancia máxima permitida entre dos segmentos consecutivos.

La principal desventaja del método de la transformada de Hough para detección de líneas resulta ser el ajuste de parámetros. El detector de bordes contiene por lo menos un parámetro ajustable de tipo de sensibilidad, la Transformada de Hough estándar involucra usualmente cuatro parámetros. Uno asociado a la escala, un segundo asociado al compromiso entre falsos positivos y falsos negativos y los otros dos a la discretización de los parámetros $r$ y $\theta$, aunque estos últimos pueden ser asociados a la resolución de la imagen. Por otro lado están los dos parámetros asociados a la etapa de extracción de segmentos [19].

El ajuste de estos parámetros puede dar problemas y desafortunadamente no hay una regla general para realizar esto. Si el ajuste es correcto este método puede proveer muy buenos resultados pero en otro caso el desempeño se puede ver fuertemente afectado.

#### 1.3.1.1. Ejemplos de interés

En la Figura **??** se muestran los resultados para el detector líneas mediante el Método de la Transformada de Hough implementado por Matlab mediante el uso de la función `houghlines`. El script utilizado se encuentra en el Centro de Documentación de Mathworks[64]. Del script proporcionado se realizaron algunos cambios, se cargó la ruta de la imagen de entrada, se eliminó la rotación a la imagen de entrada y se modificaron algunos parámetros. Los resultados mostrados representan los mejores casos para los distintos ajustes de parámetros probados. El tamaño mínimo de segmento se fijó en 15 y el máximo hueco entre segmentos a llenar en 8.



(a) El plano de Hough (b) Detección de líneas de Houhj

Figura 1.6: Resultados para la detección de líneas por el Método de Hough para los parámetos $MinLength = 15$ y $FillGap = 8$.

#### 1.3.1.2. Aplicabilidad

Se comprobó la dificultad intrínseca del método asociada al ajuste de parámetros no pudiéndose obtener los resultados deseados en ningún caso.

### 1.3.2. Detector de segmentos de línea: LSD

El algoritmo Line Segment Detector (LSD) es capaz de detectar segmentos de línea rectos y uno de los componentes principales en este proyecto. Debido a su importancia es que se le dedica el Capítulo **??** a su desarrollo.

#### 1.3.2.1. Ejemplo de interés

En la Figura 1.7 se muestran los resultados del algoritmo LSD para la imagen del marcador. Este ejemplo fue realizado mediante el uso del demo *online* de IPOL para este algoritmo[42]. Se puede observar que el algoritmo logra detectar con éxito todas las estructuras de segmento de línea presentes en la imagen y en particular las que corresponden al marcador, los lados de los cuadriláteros.

El algoritmo produjo un número de 169 segmentos de línea detectados en un tiempo de ejecución de 130*ms* sobre el servidor que aloja la aplicación. La misma ejecución realizada en el marco de este proyecto sobre una PC con procesador Intel Core 2 Duo de 2*Ghz* tomó un tiempo de aproximadamente 40*ms*.

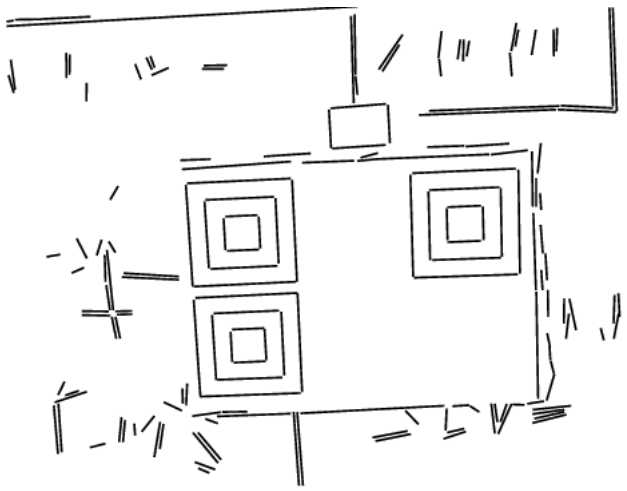


Figura 1.7: Resultados de LSD para la imagen del marcador.

### 1.3.3. Detector de segmentos de línea: EDLines

EDLines[38] es un detector de segmentos de línea de tiempo lineal que provee resultados comparables con LSD. No requiere ajuste de parámetros y se ejecuta en tiempo real. En muchos aspectos es similar al LSD ya que utiliza validación de líneas mediante el principio de Helmholtz que permite controlar el número de falsos positivos. El algoritmo hace uso del rápido detector de bordes Edge Drawing (ED) [37] desarrollado por los mismos autores que provee una cadena de píxeles borde en forma contigua.

EDLines trabaja sobre imágenes en niveles de gris y consiste básicamente en tres etapas: detección de bordes seguido por extracción de segmentos y finalmente la validación de líneas. Se describen aquí cada una de estas etapas.

- **Detección de bordes**: La detección de bordes se realiza utilizando el algoritmo Edge Drawing el cual provee resultados rápidos y libres de artefactos de ruido.

  El algoritmo se aplica a la imagen de entrada en niveles de gris. En primer lugar se le aplica un filtrado para remoción de ruido y suavizar la imagen. A continuación se calcula el gradiente, módulo y dirección, en cada pixel de la imagen suavizada. En tercer lugar se calculan un conjunto de píxeles llamados *anchors* que corresponden a puntos con alta probabilidad de ser borde. Por último se conectan estos *anchors* mediante un proceso de dibujado de bordes entre ellos basándose en el valor del módulo y dirección del gradiente previamente calculado. Este es en cierto sentido un proceso similar a los juegos de dibujo para niños.

- **Extracción de segmentos de línea**: El objetivo de esta etapa es el de "partir" la cadena contigua de píxeles obtenida mediante el detector de bordes en uno o más segmentos de línea rectos. Básicamente lo que se hace es recorrer la secuencia de píxeles e ir ajustando líneas a los píxeles utilizando el método de ajuste de líneas por mínimos cuadrados hasta que el error exceda cierto umbral. En el momento en que se excede ese umbral se genera un nuevo segmento de línea y así recursivamente se recorren todos lo píxeles de la cadena.

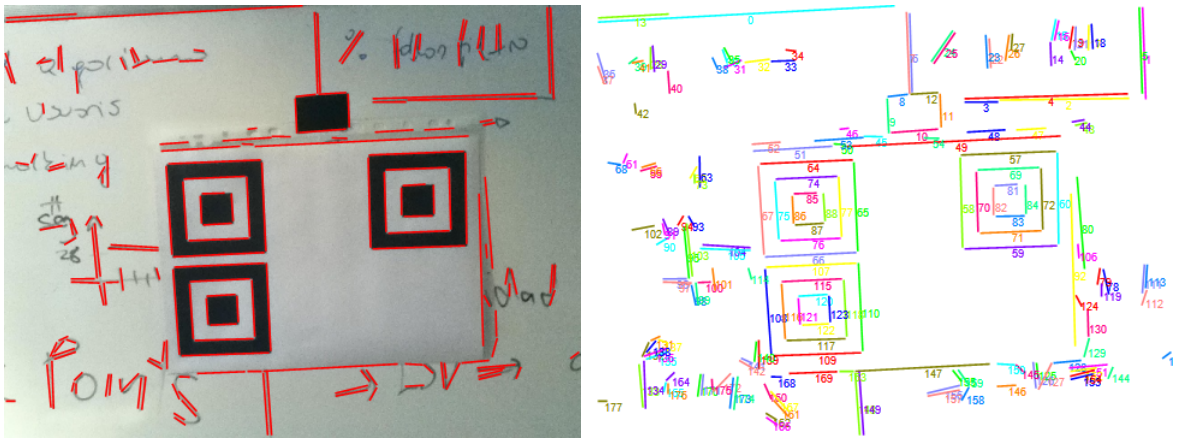
- **Validación de líneas**: El método de validación de segmentos de línea está basado en el principio de Helmholtz el cual postula que para que una estructura sea perceptualmente significativa, la esperanza de que la misma (agrupado o Gestalt) ocurra de casualidad debe ser muy baja. Este es un efoque *a contrario* en donde los objetos son detectados como *outliers* sobre el modelo de fondo. En esta etapa se logra eliminar los falsos positivos controlando el número

de falsas alarmas. Este enfoque en particular es el mismo que utiliza LSD, el cual se explica en detalle el Capítulo **??**.

Si la aplicación lo permite, esta validación puede ser eliminada o sustituida por otra, por ejemplo por un umbral de largo máximo de segmentos, acelerando aún más el algoritmo EDLines. En LSD esta etapa esta integrada con la etapa de detección de segmentos para cada iteración aunque podría ser desacoplada en cierta medida. De todas formas EDLines presenta esa ventaja de modularidad frente a LSD que puede llegar a ser útil.

### 1.3.4. Ejemplos de interés

En la Figura 1.8(a) se muestra la imagen de entrada junto con los segmentos detectados por EDLines para la imagen del marcador. En la Figura 1.8(b) se muestra únicamente los segmentos de salida de EDlines numerados. El tiempo de ejecución declarado para esta imagen por la aplicación



(a) Imagen de entrada y segmentos

(b) Segmentos de salida numerados

Figura 1.8: Resultados de EDLines para la imagen del marcador.

*on-line* es de 30*ms* en su servidor pero alega que para una PC Intel con CPU de 2*Ghz* el mismo sería de 10*ms*. Suponiendo una detección en el tiempo declarado para una PC este sería dos veces más veloz que el de LSD para la misma imagen. Se detectaron 178 segmentos y se puede ver que los resultados son muy similares a los que produce LSD que detecta 169 segmentos para la misma imagen (Figura 1.7).

#### 1.3.4.1. Aplicabilidad

El algoritmo EDLines se encuentra disponible para descarga como una librería compilada para Windows o Linux. También se puede probar en forma de demo *online* desde la página web del laboratorio Computer Vision & Pattern Recognition de la Universidad de Anadolu de Turquía en donde también explica en forma resumida su funcionamiento[67].

El código fuente por su parte no está disponible para la descarga de ningún tipo en contraste con el LSD. Esto es un factor determinante para la utilización del algoritmo en el proyecto ya que se tiene que poder compilar para la plataforma de desarrollo utilizada, en este caso sería para el sistema operativo iOS. Una implementación del algoritmo en lenguaje C/C++ no fue considerada como parte del alcance de este proyecto pero sí puede ser considerada como trabajo a futuro si se desea seguir con el enfoque de segmentos de línea para detección.

### 1.3.5. Detector de segmentos de línea y arcos: ORT

El Object Recongnition Toolkit (ORT) provee una serie de herramientas para la detección de segmentos de línea, arcos y también puntos e incluso estructuras más complejas como polígonos. El código está disponible para su descarga en la página del curso "CSE 576 Computer Vision" de la Univeristy of Washington de Estados Unidos de América[65] como también desde un repositorio SVN público de Juan Cadelino se puede encontrar otra distribución del código con documentación incluída[66].

El ORT consiste en un número de herramientas para ejecución en cascada desde consola de comandos que resultan en la detección de las características deseadas. Cada una de las herramientas corresponde a un algoritmo, estos son **Fex**[1], **Lpeg**[13] e **Ipeg**.

A continuación de explican algunas de ellas.

- **Chainpix**: produce una lista de píxeles encadenados desde una imagen binaria que es salida de un detector de bordes tipo Canny. Se ejecuta mediante

  ```
  chainpix < blocks.canny.pgm > blocks.cpx
  ```

  en donde `blocks.canny.pgm` es la imagen binaria salida de Canny y `blocks.cpx` la lista de píxeles encadenados en un formato predefinido.

- **Fex**: convierte la lista de píxeles encadenados en segmentos de línea rectos y arcos circulares. Su ejecución se realiza mediante:

  ```
  fex < blocks.cpx > blocks.fex
  ```

  y tiene como salida `blocks.fex`.

- **Lpeg**: realiza un agrupado de bajo nivel sobre los segmentos producidos por Fex hacia pares de líneas paralelas y varios tipos de junturas. Tiene ciertos parámetros opcionales como el *mínimo largo de línea*, *máximo ángulo* tolerable entre dos líneas paralelas y un *nivel de calidad* mínimo. Se ejecuta como:

  ```
  lpeg < blocks.fex > blocks.lpg
  ```

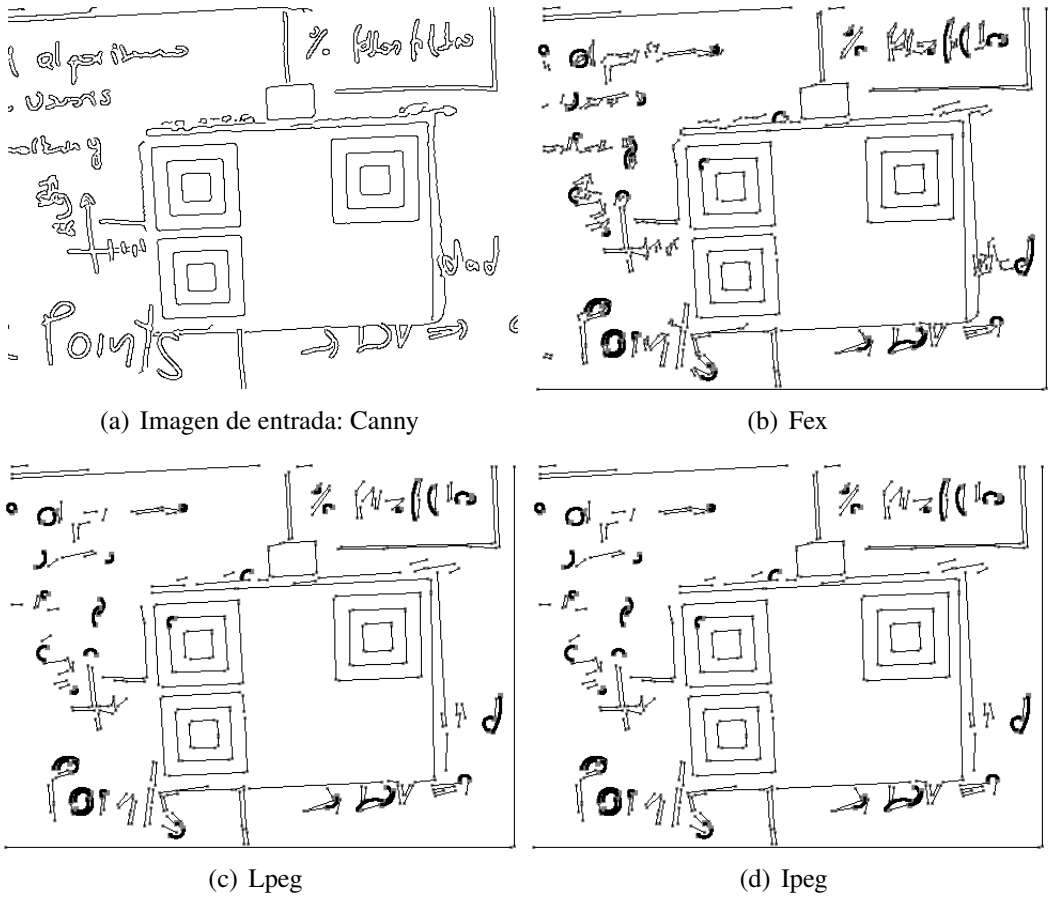
  en donde la salida se escribe en `blocks.lpg`.

- **Ipeg**: realiza un agrupado de nivel medio sobre los conjuntos producidos por Lpeg hacia agrupaciones en tipo de ternas, esquinas y polígonos. Al igual que Lpeg tiene parámetros opcionales para producir únicamente el tipo de agrupaciones de salida deseadas. Se ejecuta como:

  ```
  ipeg < blocks.lpg > blocks.ipg
  ```

  en donde la salida se escribe en `blocks.ipg`.

- **Ort2Image**: toma la salida de Fex, Lpeg o Ipeg y realiza el dibujado de segmentos de línea rectos y segmentos de arco produciendo una imagen de salida en formato PGM.



(a) Imagen de entrada: Canny (b) Fex

(c) Lpeg (d) Ipeg

Figura 1.9: Resultados del Object Recognition Toolkit (ORT) para una imagen de entrada salida de Canny.

#### 1.3.5.1. Ejemplos de interés

En la Figura 1.9 se muestran los resultados de las diferentes herramientas del ORT. Se utiliza como entrada el resultado del detector bordes de Canny para la imagen de la Figura 1.4 para $\sigma = 1{,}4$ ya que fue el valor que demostró mejores resultados entre las pruebas realizadas. Se puede observar que la salida de Fex logra capturar en buena forma los segmentos de línea correspondientes al marcador aunque alguno de ellos en forma "quebrada". También se pueden ver los segmentos de arco detectados, otro de los elementos de este algoritmo. Lpeg logra eliminar una buena parte de los segmentos dejando sólo aquellos que son en algún sentido paralelos entre sí. Por su parte Ipeg no produce ningún cambio sobre los resultados de Lpeg. En otras pruebas realizadas tampoco se lograron los resultados deseados para este algoritmo aún utilizando los parámetros opcionales. Un estudio más detallado del funcionamiento de esta herramienta en particular queda pendiente ya que resulta interesante para la aplicación.

#### 1.3.5.2. Aplicabilidad

El código fuente de estas herramientas está disponible para la descarga pero este genera un conjunto de programas ejecutables para consola de comandos. Su aplicación para el proyecto no es directa ya que se debería portar a una interfaz compatible con nuestra aplicación. Por otro lado mediante algunas pruebas realizadas sobre este algoritmo los tiempos de ejecución total no resultaron suficientemente buenos para cumplir los requerimientos de tiempo real. También se debe notar que aunque se obtuvieron resultados razonables para las herramientas Fex y Lpeg no fue así para Ipeg

que aporta un interés adicional para la aplicación.

Un estudio más detallado del código fuente y la posibilidad de re-utilizarlo dentro del marco de la aplicación queda como trabajo a futuro ya que ORT provee algunas herramientas interesantes y de más alto nivel que el detector de segmentos elegido LSD.

# CAPÍTULO 2

# Marcadores

La inclusión de *marcadores* en la escena ayuda al problema de extracción de características y por lo tanto al problema de estimación de pose [27]. 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 fácilmente utilizables para la estimación de la pose.

Los marcadores planos se pueden obtener mediante la construcción en una geometría coplanar de una serie de primitivas identificables como esquinas, segmentos o líneas. Un único marcador plano puede contener por si solo todas las seis restricciones espaciales necesarias para definir un marco de coordenadas asociado a su pose.

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.

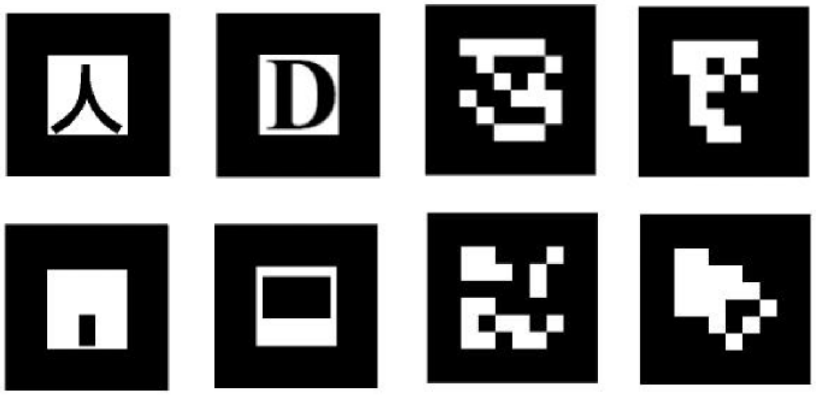
En el primer lugar se explican brevemente algunos de los sistemas de Realidad Aumentada más populares basados en marcadores planos. En segundo lugar se propone el diseño de un marcador específico para la aplicación a este proyecto y se desarrollan las soluciones a los algoritmos de detección de dicho marcador mostrando algunos resultados parciales en el proceso. Por último se muestran algunos resultados de la detección.

## 2.1. Sistemas basados en marcadores planos

Existen muchos sistemas de visión basados en *marcadores planos* con aplicación en Realidad Aumentada y Navegación. Algunos de ellos son *ARToolKit* [26], *ARTag* [14] y *ARToolKitPlus* [34] utilizados para Realidad Aumentada. A continuación se realiza una breve descripción del funcionamiento general de los mismos.

Los sistemas basados en marcadores planos utilizan típicamente marcadores bitonales. Esto perimte reducir la sensibilidad a las condiciones de luz de la escena y a las configuraciones de la cámara por lo que no hay necesidad de identificar tonos de grises y la regla de decisión para cada píxel puede ser reducida, en la versión más simple, a un umbral o *threshold* [25]. El diseño de los marcadores depende en gran medida de la aplicación. En la figura 2.1 se muestran algunos marcadores planos para aplicaciones de Realidad Aumentada en donde cada uno de ellos provee suficientes puntos para permitir el cálculo de pose tridimensional y adicionalmente contienen cierta información en su interior para permitir su identificación.

Es importante que los marcadores puedan ser localizados en un campo de visión amplio para permitir la correcta detección bajo la distorsión asociada a la transformación proyectiva que lleva el marcador en el mundo real al plano de imagen. Por otro lado, si los marcadores contienen informa-



(a) *ARToolKit* (b) *ARTag*

Figura 2.1: Cuatro ejemplos de marcadores para los sistemas de Realidad Aumentada indicados. Figura tomada de [14].

ción en su interior, esta no debe ser muy densa para permitir la recuperación de la misma a mayor distancia. Típicamente, esta información es solo una identificación entre marcadores de un mismo sistema por lo que la información es poca y esto no es un problema.

Estos sistemas contienen ciertos puntos característicos con los que se realiza el cálculo de pose. En general su contorno es basado en un cuadrilátero y se utilizan las cuatro esquinas del contorno del marcador para realizar el cálculo.

Si se utiliza un único marcador la cantidad de puntos necesarios para la estimación de pose resultan ser pocos lo que puede ser una desventaja en cuanto a la precisión del algoritmo de estimación de pose. Esto se puede mejorar construyendo un marcador más complejo compuesto de una serie de marcadores y mediante la identificación de los mismos asignar los puntos correspondientes para la estimación de pose.

### 2.1.1. ARToolKit

*ARToolKit* es un muy popular sistema de marcadores planos para Realidad Aumentada e Interacción Hombre-Computador. Su popularidad reside ser de los primeros proyectos en utilizar la Realidad Aumentada en dispositivos móviles y también debido a que es un proyecto de código abierto.

Los marcadores bitonales consisten en un cuadrado con borde negro y un patrón en el interior. La primer etapa del proceso de reconocimiento consisten en detectar los bordes negros. Esto se realiza buscando grupos conexos de píxeles (*blobs*) que están por debajo de un determinado umbral. Posteriormente se extrae el contorno de cada grupo esos grupos que están rodeados por cuatro líneas rectas son marcados como marcadores potenciales. Las cuatro esquinas de cada marcador potencial son utilizados para calcular la homografía y así remover la distorsión perspectiva. Con el marcador en una vista canónica, se procede a identificar el patrón interno muestreando en una grilla, de por lo general $16 \times 16$ o $32 \times 32$, los valores de gris. Con esto se construye un vector característico y se compara por correlación con una librería de vectores de característicos logrando la identificación del mismo.

Este sistema tiene algunas desventajas o "puntos débiles". En primer lugar la detección es basada en un umbral por lo que las condiciones de iluminación pueden afectar fuertemente la efectividad de la misma. Dado que el código esta disponible este se puede modificar para realizar *threshold* local o adaptivo por ejemplo. Otras desventajas están relacionadas con el proceso de identificación

del marcador frente a la librería.

### 2.1.2. ARTag

*ARTag* es otro sistema de marcadores planos para Realidad Aumentada e Interacción Hombre-Computador que surge como una evolución de ciertos aspectos de *ARToolKit*. Los marcadores son también bitonales y basados en un borde negro. En contraste con el *ARToolKit* este sistema utiliza un enfoque basado en bordes por lo que es más robusto a condiciones de iluminación. Los bordes son unidos en segmentos que a su vez se unen en cuadriláteros. Al igual que con *ARToolKit* con las esquinas se calcula la homografía y se muestrea en el interior del marcador pero con una grilla de $6 \times 6$.

El sistema puede lidiar con condiciones de iluminación cambiantes, oclusiones y segmentos partidos hasta cierto punto. Otra mejora notable con respecto a *ARToolKit* reside en el sistema de identificación de marcadores. El proceso de identificación de los marcadores entre sí es a su vez más veloz y robusto que el de *ARToolKit*.

## 2.2. Marcador QR

El enfoque elegido para la detección de características utilizando marcadores parte del trabajo de fin de curso denominado Autoposicionamiento 3D de *Matías Tailanián* y *Santiago Paternain* para el curso *Tratamiento de imágenes por computadora* de Facultad de Ingeniería, Universidad de la República[32]. 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[**?**] por su buena *performance*.

El marcador utilizado está basado en la estructura de detección incluida en los códigos *QR* y se muestra en la figura 2.2. Éste consiste en tres grupos idénticos de tres cuadrados concéntricos superpuestos en "capas". La primer capa contiene el cuadrado negro de mayor tamaño, en la segunda capa se ubica el cuadrado mediano en color blanco y en la última capa un cuadrado negro pequeño. De esta forma se logra un fuerte contraste en los lados de cada uno de los cuadrados facilitando la detección de bordes o líneas. El resultado de una detección de líneas para esta configuración produce para cada cuadrado la detección de sus lados. A diferencia de los códigos *QR* la disposición espacial de los grupos de cuadrados es distinta para evitar ambigüedades en la identificación de los mismos entre sí.

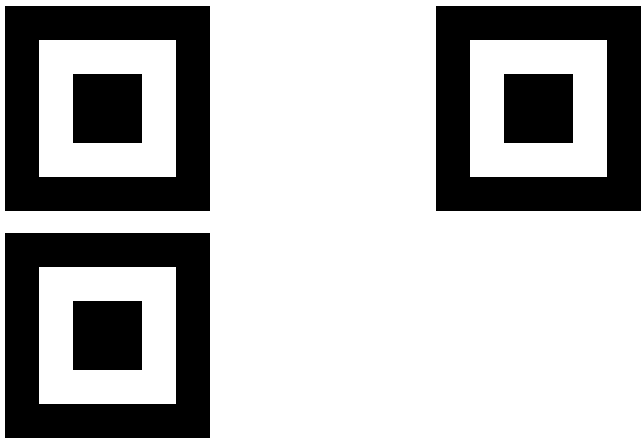


Figura 2.2: Marcador propuesto basado en la estructura de detección de códigos QR.

### 2.2.1. Estructura del marcador

A continuación se presentan algunas definiciones de las estructuras básicas que componen 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 más 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* y se construye a partir de $M$ cuadriláteros, con $M > 1$. Los cuadriláteros comparten un mismo centro pero se diferencian en un factor de escala. 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*, $\mathbf{c}_i$, 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 de 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 disposición espacial 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 para proveer las correspondencias $\mathbf{M}_i \leftrightarrow \mathbf{m}_i$ al algoritmo de estimación de pose. De esta forma se tienen una cantidad de puntos superior a los que se podrían tener utilizando uno de los marcadores de los sistemas como *ARToolKit* a un costo de detección relativamente bajo. Por otro lado se podría agregar algún patrón para la identificación de marcadores en la esquina que completa el rectángulo en donde no hay *QlSet* como se realizó en el trabajo Autoposicionamiento 3D [32].

### 2.2.2. 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 más específicamente en tres conjuntos de tres cuadrados dispuestos en como se muestra en la figura 2.2.

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 2.3. 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 área de Procesamiento 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 2.4.

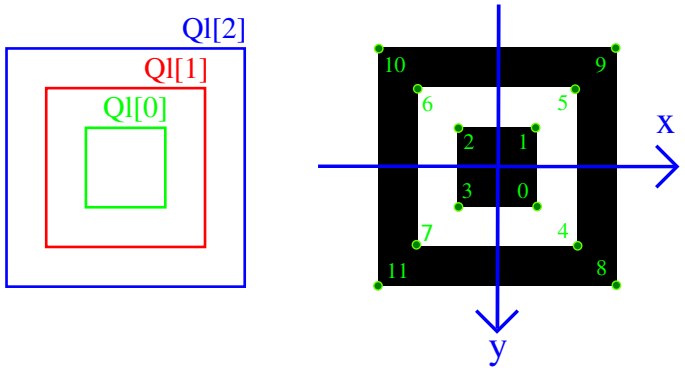



Figura 2.3: Detalle de un *QlSet*. A la izquierda se muestra el resultado de la detección de un *QlSet* y el orden interno de sus cuadriláteros y a la derecha el orden de los vértices respecto al sistema de coordenadas local.

Un detalle del marcador completo se muestra en la figura 2.4 en donde se define el conjunto $i$ de cuadriláteros concéntricos como el $QlSet[i]$ y se definen los respectivos centros de cada uno de ellos como $\mathbf{c}_i$. 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 *QlSet* entre sí basados únicamente en la posición de sus centros y es explicado en la sección de determinación de correspondencias (sec.: 2.3.3).

### 2.2.3. Parámetros de diseño

Provisto el diseño del marcador descrito, quedan definidos ciertos parámetros **estructurales** que fueron de tomados fijos a lo largo del proyecto pero que podrían ser cambiados para 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 ($Ql$).

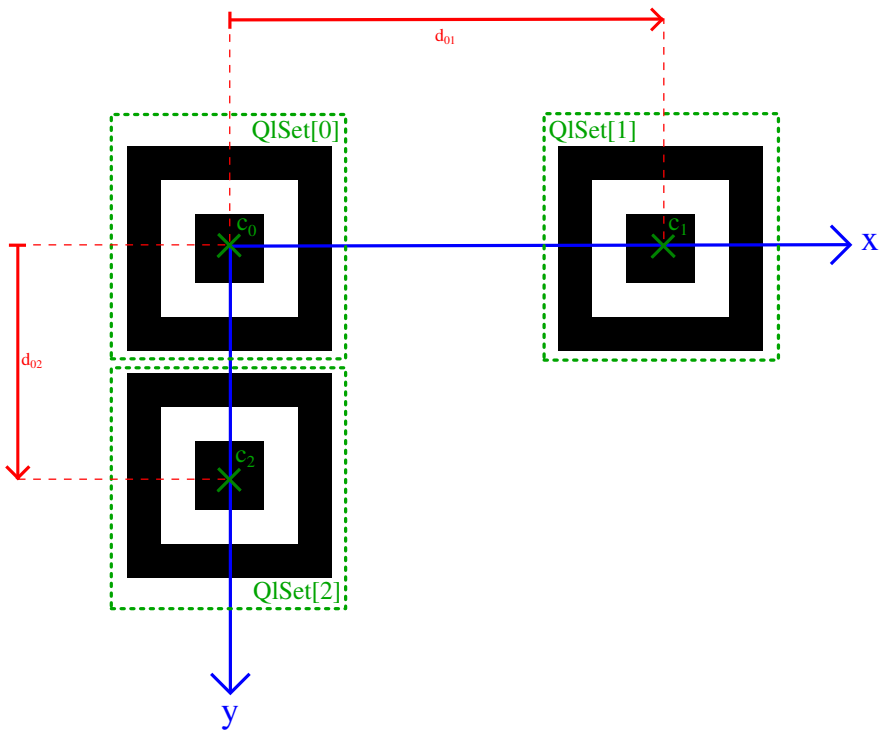



Figura 2.4: Detalle del marcador propuesto formando un sistema de coordenadas.

- Disposición: disposición espacial de los conjuntos de cuadriláteros (*QlSet*).

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 tiene 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 simultáneamente en los segmentos verticales como en los horizontales. De otra forma se tendría una dirección que limita más 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 2.4.

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 más 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.

### 2.2.4. Diseños utilizados

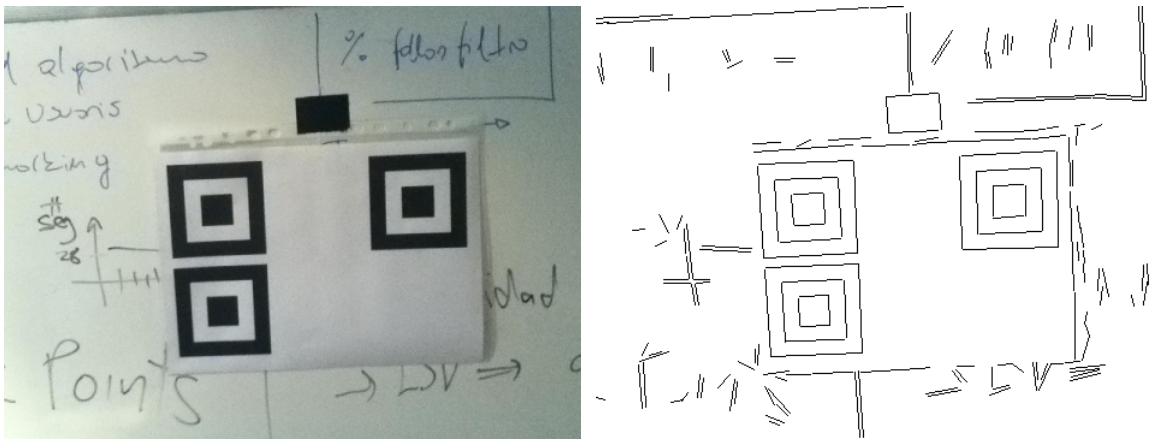
- **Test**: Durante el desarrollo de los algoritmos de detección e identificación de los vértices del marcador QR se trabajó con determinados parámetros de diseño de dimensiones apropiadas para posibilitar el traslado y las pruebas domésticas.
  - $l = 30mm$
  - $d_{01} = 190mm$
  - $d_{02} = 100mm$
- **Da Vinci**
- **Artigas**
- **Mapa**

## 2.3. 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.

### 2.3.1. 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.



(a) Entrada: imagen conteniendo al marcador (b) Salida: segmentos de línea detectados por LSD

Figura 2.5: Resultados del algoritmo de detección de segmentos de línea LSD.

### 2.3.2. Filtrado y agrupamiento de segmentos

El filtrado y agrupamiento de segmentos consiste en la búsqueda de conjuntos de cuatro segmentos conexos en la lista de segmentos de línea detectados por LSD. Los conjuntos de segmentos conexos encontrados se devuelven en una lista en el mismo formato a la de LSD pero agrupados de a cuatro. 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 una primera etapa $\mathbf{s}_j$ y en una segunda etapa $\mathbf{s}_k$, de forma que se forme una "U" como se muestra en la figura 2.6. La tercer etapa de búsqueda consiste en completar ese conjunto con un cuarto segmento $\mathbf{s}_l$ que cierre la "U".

Dos segmentos $\mathbf{s}_i$ y $\mathbf{s}_j$ son vecinos si se cumple que la distancia euclídea 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 segmento 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 aún más restringida probando para el segmento $\mathbf{s}_l$ correspondencia simultánea entre sus puntos y solamente un punto cada uno de los segmentos $\mathbf{s}_j$ y $\mathbf{s}_k$ .

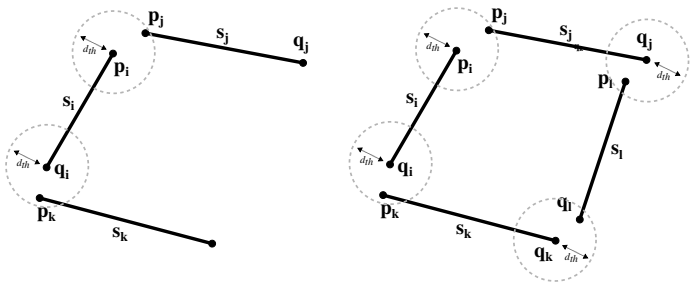
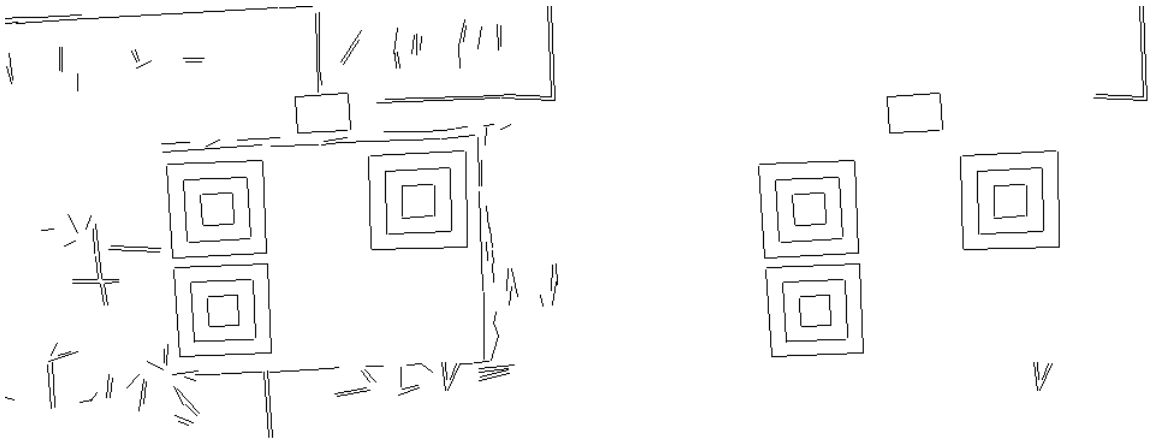



Figura 2.6: Conjunto de cuadriláteros conexos. A la izquierda la primera y segunda etapa del filtrado completadas para el segmento $\mathbf{s}_i$ en donde se busca una "U". A la derecha la última etapa en donde se cierra la "U" con el segmento $\mathbf{s}_l$.

Una vez encontrado el conjunto de cuatro segmentos conexos estos se marcan 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.

En la figura 2.3.2 se muestran lo resultados obtenidos para el algoritmo tomando como entrada la lista de segmentos de LSD. Se puede ver que los lados de los cuadrados del marcador son detectados correctamente pero también hay otras detecciones presentes. Por ejemplo el rectángulo negro correspondiente a un trozo de cinta negra que sostenía el marcador (ver figura **??**(a)). También sobreviven otro tipo de elementos indeseados que se explican a continuación.

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



(a) Entrada: segmentos de línea detectados por LSD (b) Salida: segmentos de línea filtrados y agrupados

Figura 2.7: Resultados del algoritmo de filtrado y agrupamiento de segmentos de línea.

La detección de falsos positivos se puede atribuir principalmente a la condición de vecindad utilizada en donde un caso como el que se muestra en la figura 2.8 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 el punto de intersección entre los segmentos y la distancia de este punto a los puntos **p**, **q** más cercanos de cada segmento. Como se explicará en le sección **??**, debido a que el algoritmo de determinación de correspondencias realiza la intersección entre estos segmentos se puede chequear alguna condición sobre los segmentos o su intersección y en ese momento filtrar estos casos.

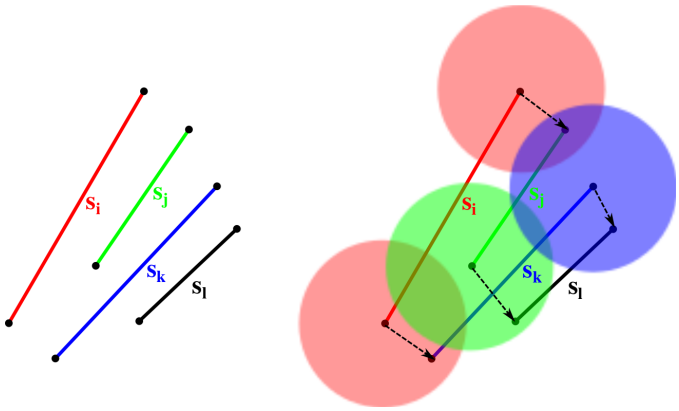



Figura 2.8: Posible configuración de segmentos paralelos que "sobreviven" al filtrado. A la izquierda el grupo de segmentos, a la derecha se muestra como se desarrolla el filtrado de $\mathbf{s}_i$.

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 resulta en buenos resultados para la aplicación pero en ciertas circunstancias produce este problema. Esta medida de distancia se podría tomar relativa al largo del los segmentos a *testear* de forma de generalizar el valor pero se debería analizar un poco más en detalle la posible implementación para que resulte en buenos resultados y no introduzca otra clase de errores.

El algoritmo de filtrado y agrupamiento de segmentos es sensible respecto a la elección del parámetro $d_{th}$. Si esta parámetro esta por debajo del valor óptimo la *performance* del algoritmo

se vera afectada fuertemente pues se corre el riesgo de sobre filtrar y no proporcionar suficientes segmentos para la correcta determinación de correspondencias. Por el contrario, si el parámetro esta por encima del valor óptimo, el filtrado tiende a proveer falsos positivos aun que este caso no llega a ser tan crítico como el primero para la aplicación. A modo de ejemplo, para una imagen de tamaño $480 \times 320$ con el marcador ocupando entre un 25 % y un 80 % el valor del parámetro que da mejores resultados es aproximadamente de 6 o 7 píxeles.

### 2.3.3. 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.: 2.2.1), de forma de desarrollar un algoritmo modular, escalable y simple.

Se toma como entrada la lista de segmentos filtrados y agrupados

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

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 $n$ 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 2.9 en donde una de ellas tiene sentido horario y la otra antihorario partiendo de $v_i$.

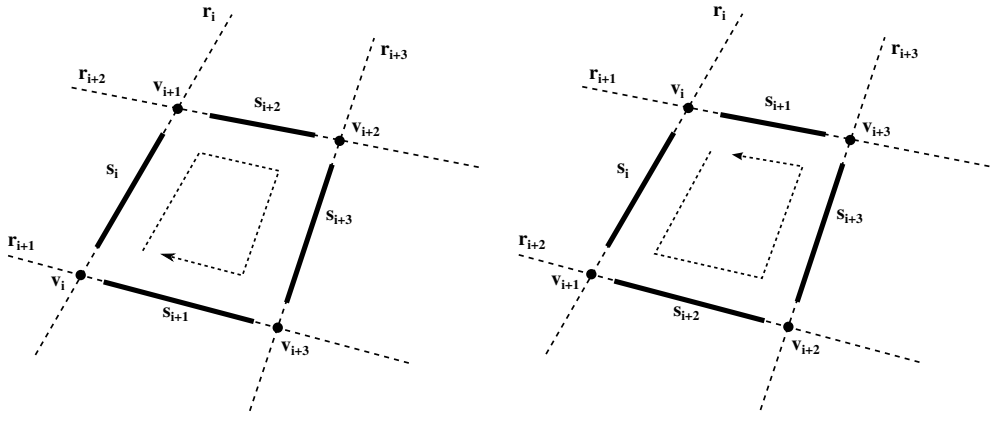



Figura 2.9: Posibles configuraciones de vértices posterior a la intersección de conjuntos de segmentos pertenecientes a un cuadrilátero.

Posterior a la intersección se realiza un chequeo sobre el valor de las coordenadas de los 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 filtrado de "falsos cuadriláteros" corrigiendo

un defecto del filtrado de segmentos, como por ejemplo un grupo de segmentos paralelos cercanos como ya se explicó.

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 $c_{th}$,

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

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 búsqueda 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 disposición 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 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 "mucho mayor" se asegura 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 2.10.

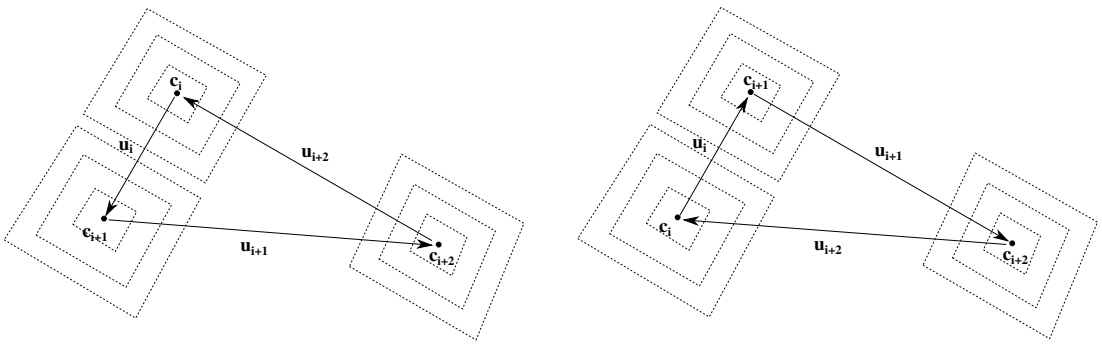



Figura 2.10: Vértices de cada *Ql* ordenados respecto al signo de sus proyecciones contra el sistema de coordenadas local a cada *QlSet*.

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 **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 contrario 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 la ecuación 2.1. Los ejes de este sistema de coordenadas permiten, para cada *Ql* de cada *QlSet*, proyectar los vértices sobre el sistema de coordenadas local al *QlSet* y según su signo ordenarlos como se muestra en la figura 2.11. De esta forma, recorriendo ordenada-

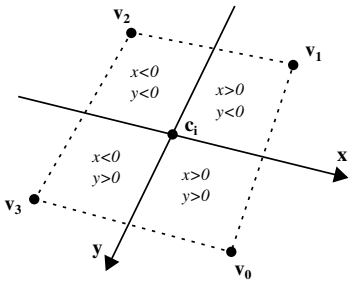



Figura 2.11: Posibles configuraciones de centros resultan en la orientación de los vectores $\mathbf{u}_{i+k}$.

mente los elementos del marcador, se ordenan los vértices de cada *Ql* del marcador.

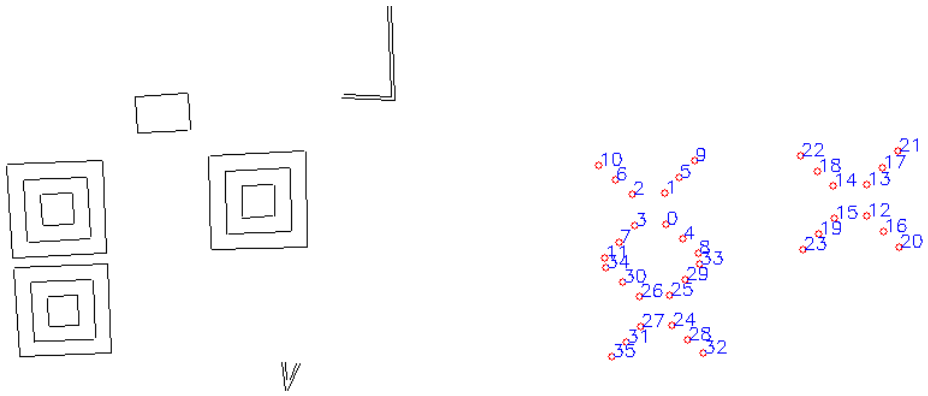
Por último, a partir del marcador ordenado, se extrae una lista de vértices que se corresponde con la lista de vértices del marcador en coordenadas del mundo. Este recorrido se realiza en el siguiente orden,

Se determinan las correspondencias $\mathbf{M}_i \leftrightarrow \mathbf{m}_i$ necesarias para la estimación de pose las cuales se muestran en la figura 2.3.3. Se puede ver que el algoritmo de determinación de correspondencias funciona correctamente por lo que los "falsos" cuadriláteros que sobreviven al filtrado de segmentos no son un problema.

### 2.3.4. Detección robusta

El algoritmo 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

**for** $i = (0,1,2)$ **do**
  **for** $j = (0,1,2)$ **do**
    **for** $k = (0,1,2,3)$ **do**
      So obtiene el punto vértice: $\mathbf{p} = QlSet[i] \rightarrow Ql[j] \rightarrow v[k]$;
      Se agrega a la lista de correspondencias $\mathbf{m}_l \leftarrow \mathbf{p}$;
      Se incrementa $l$;
    **end**
  **end**
**end**



(a) Entrada: segmentos de línea filtrados y agrupados

(b) Salida: puntos vértices ordenados.

Figura 2.12: Resultados del algoritmo de determinación de correspondencias.

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 la mayor limitante, la fluidez visual dada por el *frame rate* se ve notablemente perjudicada resultando en que el sistema sea incómodo 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 más 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 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 ordenando 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.

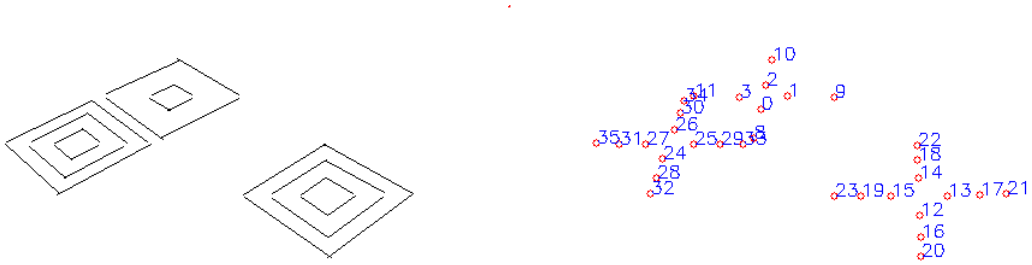
La implementación de esta extensión del algoritmo se realizó manteniendo la estructura básica descrita anteriormente y se detalla aquí solamente los agregados realizados.

Al realizar la búsqueda 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 al menos uno de ellos detectado completo, se 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.

En la figura 2.3.4(a) se muestran imagen en la que falla el filtrado de segmentos para uno de los cuadriláteros mientras que en la figura 2.3.4(b) se puede ver como el algoritmo de determinación de correspondencias provee 32 correspondencias ordenadas correctamente, diferenciando en el *QlSet* incompleto los vértices.




(a) Entrada: segmentos de línea filtrados y agrupados

(b) Salida: puntos vértices ordenados.

Figura 2.13: Resultados del algoritmo de determinación de correspondencias robusto para una falla en el filtrado de segmentos.

### 2.3.5. Resultados

más imágenes con resultados?
Todo sobre la misma imagen de entrada?
O toda la secuencia para distintas imágenes?

# 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.