

---

# Índice general

|  |          |
|--|----------|
| <b>Índice general</b>                                    | <b>1</b> |
| <b>1. Hardware y Software</b>                            | <b>4</b> |
| 1.1. Introducción . . . . .                              | 4        |
| 1.2. Software de procesamiento de imágenes . . . . .     | 4        |
| 1.3. Dispositivos móviles . . . . .                      | 4        |
| 1.3.1. iPhone . . . . .                                  | 4        |
| 1.3.2. iPad . . . . .                                    | 5        |
| 1.3.3. iPod Touch . . . . .                              | 5        |
| 1.4. Entorno de desarrollo . . . . .                     | 5        |
| 1.4.1. xCode y Objective-C . . . . .                     | 5        |
| 1.4.2. Simulador . . . . .                               | 5        |
| 1.4.3. Instruments . . . . .                             | 5        |
| 1.4.3.1. Time Profiler . . . . .                         | 5        |
| 1.4.3.2. Memory Leak . . . . .                           | 5        |
| 1.4.4. Librerías . . . . .                               | 5        |
| 1.4.4.1. AvFoundation . . . . .                          | 5        |
| 1.4.4.2. MediaPlayer . . . . .                           | 5        |
| 1.4.4.3. CoreMotion . . . . .                            | 5        |
| 1.4.4.4. Tweeter y mensajería . . . . .                  | 5        |
| 1.5. Herramientas . . . . .                              | 5        |
| <b>2. Detección</b>                                      | <b>6</b> |
| 2.1. Tipos de características . . . . .                  | 6        |
| 2.2. Bordes y esquinas . . . . .                         | 6        |
| 2.2.1. Detector de bordes de Canny . . . . .             | 6        |
| 2.2.2. Detector de bordes y esquinas de Harris . . . . . | 6        |
| 2.2.3. SUSAN Y FAST . . . . .                            | 6        |
| 2.3. Líneas y segmentos de línea . . . . .               | 6        |
| 2.3.1. Detector de líneas de Hough . . . . .             | 6        |
| 2.3.2. Detector de segmentos de línea: LSD . . . . .     | 6        |
| 2.4. Regiones y puntos de interés . . . . .              | 6        |
| 2.4.1. FAST . . . . .                                    | 6        |
| 2.4.2. Blobs . . . . .                                   | 6        |
| 2.5. Detección sin primitivas markerless . . . . .       | 6        |
| 2.6. Marcadores . . . . .                                | 7        |
| 2.7. Marcador QR . . . . .                               | 7        |
| 2.7.1. Estructura del marcador . . . . .                 | 8        |

|           |   |           |
|-----------|---|-----------|
| 2.7.2.    | Diseño . . . . .  | 8         |
| 2.7.2.1.  | Parámetros de diseño . . . . .  | 9         |
| 2.7.2.2.  | Diseño <i>test</i> . . . . .  | 11        |
| 2.7.2.3.  | Diseño <i>Da Vinci</i> . . . . .  | 11        |
| 2.7.2.4.  | Diseño <i>Artigas</i> . . . . .   | 11        |
| 2.7.2.5.  | Diseño <i>Mapa</i> . . . . .  | 11        |
| 2.7.3.    | Detección . . . . .   | 11        |
| 2.7.3.1.  | Detección de segmentos de línea . . . . .                                 | 11        |
| 2.7.3.2.  | Filtrado de segmentos . . . . .   | 11        |
| 2.7.3.3.  | Determinación de correspondencias . . . . .                               | 12        |
| 2.7.3.4.  | Resultados . . . . .  | 12        |
| <b>3.</b> | <b>LSD: “Line Segment Detection”</b>                                      | <b>13</b> |
| 3.1.      | Introducción . . . . .  | 13        |
| 3.2.      | <i>Line-support regions</i> . . . . .                                     | 13        |
| 3.3.      | Aproximación de las regiones por rectángulos . . . . .                    | 14        |
| 3.4.      | Validación de segmentos . . . . .   | 15        |
| 3.5.      | Refinamiento de los candidatos . . . . .                                  | 16        |
| 3.6.      | Optimización del algoritmo para tiempo real . . . . .                     | 16        |
| 3.6.1.    | Filtro Gaussiano . . . . .  | 17        |
| 3.6.2.    | <i>Level-line angles</i> . . . . .  | 19        |
| 3.6.3.    | Refinamiento y mejora de los candidatos . . . . .                         | 19        |
| 3.6.4.    | Algoritmo en precisión simple . . . . .                                   | 19        |
| 3.6.5.    | Resultados . . . . .  | 20        |
| 3.6.5.1.  | Filtro Gaussiano . . . . .  | 20        |
| 3.6.5.2.  | <i>Line Segment Detection</i> . . . . .                                   | 20        |
| <b>4.</b> | <b>Modelo de cámara y estimación de pose</b>                              | <b>23</b> |
| <b>5.</b> | <b>Estimación de pose monocular</b>                                       | <b>24</b> |
| 5.1.      | Calibración de cámara: modelo pin-hole [?] . . . . .                      | 24        |
| 5.1.1.    | Fundamentos y definiciones . . . . .                                      | 24        |
| 5.1.2.    | Matriz de proyección . . . . .  | 26        |
| <b>6.</b> | <b>POSIT: <i>POS</i> with <i>ITerations</i></b>                           | <b>30</b> |
| 6.1.      | Introducción . . . . .  | 30        |
| 6.2.      | POSIT clásico . . . . .   | 30        |
| 6.2.1.    | Notación y definición formal del problema de estimación de pose . . . . . | 30        |
| 6.2.2.    | SOP: Scaled <i>Ortographic Projection</i> . . . . .                       | 32        |
| 6.2.3.    | Ecuaciones para calcular la proyección perspectiva . . . . .              | 32        |
| 6.2.4.    | Algoritmo . . . . .   | 33        |
| 6.3.      | POSIT moderno . . . . .   | 33        |
| 6.4.      | SoftPOSIT . . . . .   | 33        |
| 6.5.      | POSIT Coplanar . . . . .  | 33        |
| <b>7.</b> | <b>Implementación</b>   | <b>34</b> |
| 7.1.      | Introducción . . . . .  | 34        |
| 7.2.      | Diagrama global de la aplicación . . . . .                                | 34        |
| 7.3.      | TableViewCellController . . . . .   | 35        |

|          |  |    |
|----------|--|----|
| 7.4.     | QR . . . . .   | 35 |
| 7.4.1.   | QR. Una realidad . . . . .                                       | 35 |
| 7.4.2.   | Qué son realmente los QRs? . . . . .                             | 35 |
| 7.4.3.   | Codificación y decodificación de QRs . . . . .                   | 36 |
| 7.4.4.   | El QR en la aplicación . . . . .                                 | 37 |
| 7.4.5.   | Arte con QRs . . . . .   | 37 |
| 7.5.     | Servidor . . . . .   | 37 |
| 7.5.1.   | Creando el servidor . . . . .                                    | 38 |
| 7.5.1.1. | Servidor iOS . . . . .   | 38 |
| 7.5.1.2. | Servidor LAMP . . . . .  | 38 |
| 7.5.2.   | Lenguaje php y principales scripts . . . . .                     | 39 |
| 7.6.     | SIFT . . . . .   | 39 |
| 7.7.     | Incorporación de la realidad aumentada a la aplicación . . . . . | 39 |

---

# 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 tamaño de la pantalla entre otras puedan verse afectadas. A continuación se relata el surgimiento de cada uno de los 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 años. 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 desbloquear 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 años. 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 milímetros, 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. Tipos 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 [?]. 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.

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. El enfoque elegido

## 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<sup>1</sup>. 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 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 tres grupos idénticos de tres cuadrados concéntricos superpuestos de tal forma que los lados de cada uno de tres cuadrados son visualizables. A diferencia de los códigos *QR* la disposición de los grupos de cuadrados es distinto para evitar ambigüedades en la determinación de su posicionamiento espacial. Estas dos características son esenciales para la extracción de los *puntos fiduciales* de forma coherente, es decir, las correspondencias tienen que poder ser determinadas completamente bajo criterios razonables.

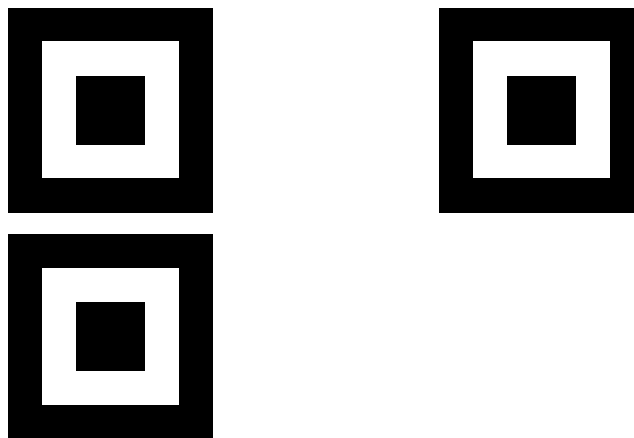


Figura 2.1: Marcador propuesto basado en la estructura de detección de códigos QR.

<sup>1</sup> 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 ( $Ql[0], Ql[1], \dots, Ql[M-1]$ ) 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 ( $QlSet[0], QlSet[1], \dots, QlSet[N-1]$ ) 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. Las direcciones de los ejes son muy utilizadas en el ambiente de las imágenes para definir los ejes de una imagen. Definido el sistema de coordenadas de puede fijar un orden a los *vértices*  $v_{j_i}$  de cada *cuadrilátero*  $Ql[j]$  como,



$$v_{j_0} = (a/2, a/2) \quad v_{j_1} = (a/2, -a/2) \quad (2.1)$$

$$v_{j_2} = (-a/2, -a/2) \quad v_{j_3} = (-a/2, a/2) \quad (2.2)$$

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.

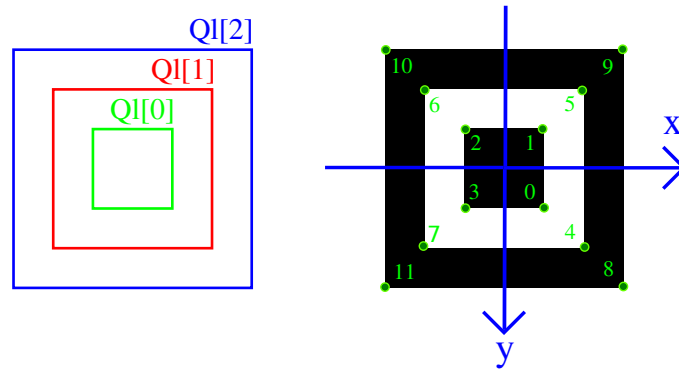


Figura 2.2: Detalle de un QISet. Orden de los *cuadriláteros* a la izquierda. Orden de los *vértices* a la derecha.

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

$$\mathbf{x} = \frac{\mathbf{c}_1 - \mathbf{c}_0}{\|\mathbf{c}_1 - \mathbf{c}_0\|} \quad \mathbf{y} = \frac{\mathbf{c}_2 - \mathbf{c}_0}{\|\mathbf{c}_2 - \mathbf{c}_0\|} \quad (2.3)$$

La disposición de los QISet 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 QISet 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 tomados fijos a lo largo del proyecto pero que podrían ser 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 (QI).
- Disposición: disposición espacial de los *conjuntos de cuadriláteros* (QISet).

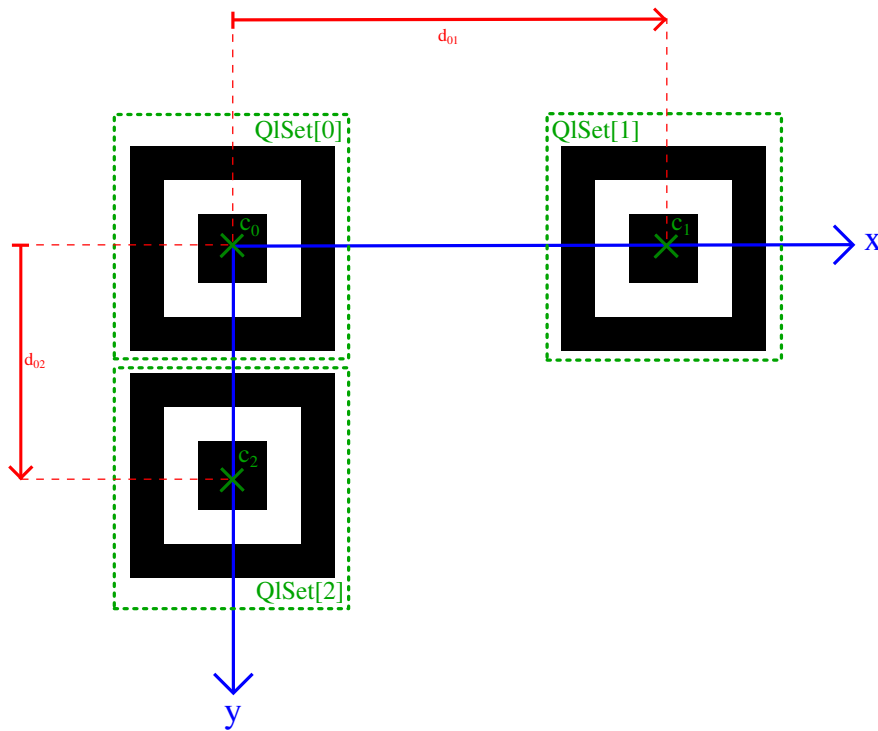


Figura 2.3: Detalle del marcador propuesto formando un sistema de coordenadas.

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  $3 \times \text{QISet}$ 's conteniendo  $3 \times \text{QI}$ '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 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.

Por otro lado se tiene otro juego de parámetros 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*  $\text{QISet}[j]$  con  $\text{QISet}[i]$ .
- $l$ : lado del *cuadrilátero* mas pequeño ( $\text{QI}[0]$ ) de los  $\text{QISet}$ .

En este caso se debe siempre cumplir 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. Diseño *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.

### 2.7.2.3. Diseño *Da Vinci*

### 2.7.2.4. Diseño *Artigas*

### 2.7.2.5. Diseño *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 ??).

### 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  $m \times n$  y devuelve una lista de segmentos en forma de pares de puntos de origen y destino.

```
int nb;
double *img;
...
out = lsd(&nb, img, m, n);
```

### 2.7.3.2. Filtrado de segmentos

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

Se parte de una lista de  $n$  segmentos de línea y se recorre la lista de segmentos en busca de segmentos vecinos. Para el  $i$ -ésimo segmento se testean sus dos puntos  $\mathbf{p}_i$  y  $\mathbf{q}_i$  con los puntos del  $j$ -ésimo segmento  $\mathbf{p}_j$  y  $\mathbf{q}_j$  para  $j = (i + 1, \dots, n)$ . Dos segmentos son vecinos si se cumple que la distancia entre puntos,  $d_{ij}$ , es menor a un cierto umbral para alguna de las combinaciones.

**Algorithm 1:** filterSegments**Data:** list= $\{(\mathbf{p}_0, \mathbf{q}_0), (\mathbf{p}_1, \mathbf{q}_1), \dots, (\mathbf{p}_{n-1}, \mathbf{q}_{n-1})\}$ .**Result:** filteredlist= $\{(\mathbf{p}_0, \mathbf{q}_0), (\mathbf{p}_1, \mathbf{q}_1), \dots, (\mathbf{p}_{m-1}, \mathbf{q}_{m-1})\}$ . Lista ordenada.**begin** $V \leftarrow U;$  $S \leftarrow \emptyset;$ **for**  $x \in X$  **do** $NbSuccInS(x) \leftarrow 0;$  $NbPredInMin(x) \leftarrow 0;$  $NbPredNotInMin(x) \leftarrow |ImPred(x)|;$ **for**  $x \in X$  **do****if**  $NbPredInMin(x) = 0$  **and**  $NbPredNotInMin(x) = 0$  **then** $\quad AppendToMin(x)$ 

1

**while**  $S \neq \emptyset$  **do**

REM

 $\quad$  remove  $x$  from the list of  $T$  of maximal index;

2

 $\quad$  **while**  $|S \cap ImSucc(x)| \neq |S|$  **do** $\quad\quad$  **for**  $y \in S - ImSucc(x)$  **do** $\quad\quad\quad$  { remove from  $V$  all the arcs  $zy :$  }; $\quad\quad\quad$  **for**  $z \in ImPred(y) \cap Min$  **do** $\quad\quad\quad\quad$  remove the arc  $zy$  from  $V$ ; $\quad\quad\quad\quad$   $NbSuccInS(z) \leftarrow NbSuccInS(z) - 1;$  $\quad\quad\quad\quad$  move  $z$  in  $T$  to the list preceding its present list; $\quad\quad\quad\quad$  {i.e. If  $z \in T[k]$ , move  $z$  from  $T[k]$  to  $T[k - 1]$ }; $\quad\quad\quad$   $NbPredInMin(y) \leftarrow 0;$  $\quad\quad\quad$   $NbPredNotInMin(y) \leftarrow 0;$  $\quad\quad\quad$   $S \leftarrow S - \{y\};$  $\quad\quad\quad$   $AppendToMin(y);$  $\quad\quad$   $RemoveFromMin(x);$ **end****2.7.3.3. Determinación de correspondencias****2.7.3.4. Resultados**

---

## 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 precisión 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 de 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  $\tau$ , 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 incluidos 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.

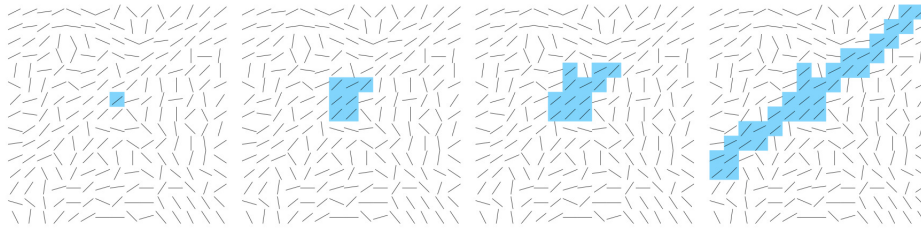


Figura 3.1: Proceso de crecimiento de una región. El ángulo asociado cada píxel de la imagen está representado por los pequeños segmentos y los píxeles coloreados representan la formación de la región. Fuente [?].

Los píxeles 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  $\tau$  descrita 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 [?].

Sin embargo, LSD plantea un método para ahorrarse 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 pudiese 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 cubrir el 99% de la masa de la región.

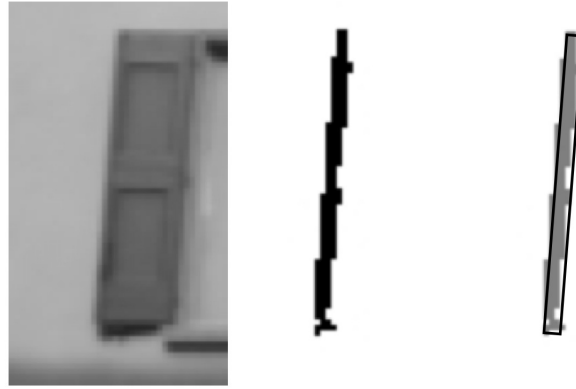


Figura 3.3: Búsqueda del segmento que mejor aproxime cada *line-support region*: aproximación de una región por un rectángulo. Izq.: Imagen original. Ctro.: Una de las regiones computadas. Der.: Aproximación rectangular que cubre el 99 % de la masa de la región. Fuente [?].

### 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 distribuidos 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  $X$ :

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

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

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

donde empíricamente  $\varepsilon = 1$  para todos los casos.

Si se toma en cuenta que cada píxel de la imagen ruidosa toma un valor independiente de los demás, se concluye que también lo harán su gradiente y su *level-line orientation field*. De esta manera, dada una orientación aleatoria cualquiera, la probabilidad de que uno de los píxeles de la imagen cuente

con dicha orientación, a menos de la ya mencionada tolerancia  $\tau$ , 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 imagen analizada de la siguiente manera:

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

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

### 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 disminuir 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  $\tau$ . 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 algoritmos 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, siempre 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, disminuir el tamaño de la imagen de entrada con el objetivo de disminuir el volumen de información procesada. Además, al difuminar la imagen, se conservan únicamente los bordes más pronunciados. Para este proyecto en particular, se escogió la escala del submuestreo fija en 0,5, un poco más adelante en la corriente sección se explicará por qué.

El filtrado de la imagen se hace en dos pasos, primero a lo ancho y luego a lo largo. Se utiliza el núcleo Gaussiano normalizado de la figura 3.4.

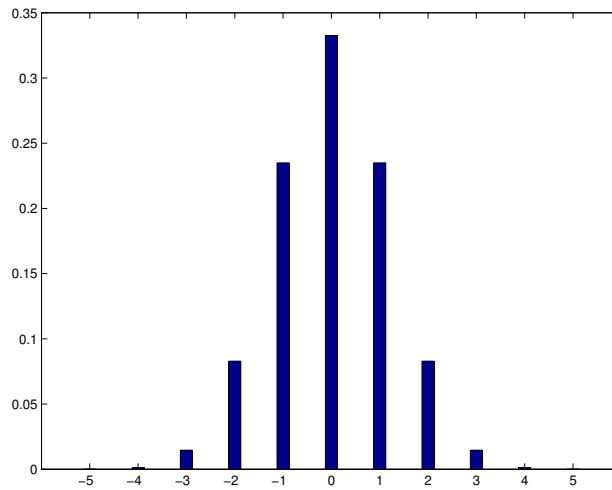


Figura 3.4: Núcleo Gaussiano utilizado por LSD.  $\sigma = 1, 2$ .

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 promedio de los píxeles de la imagen original (y luego la auxiliar) sea la debida. Aunque esta operación le agrega precisión al algoritmo, también le agrega un gran costo computacional, ya que lo que se

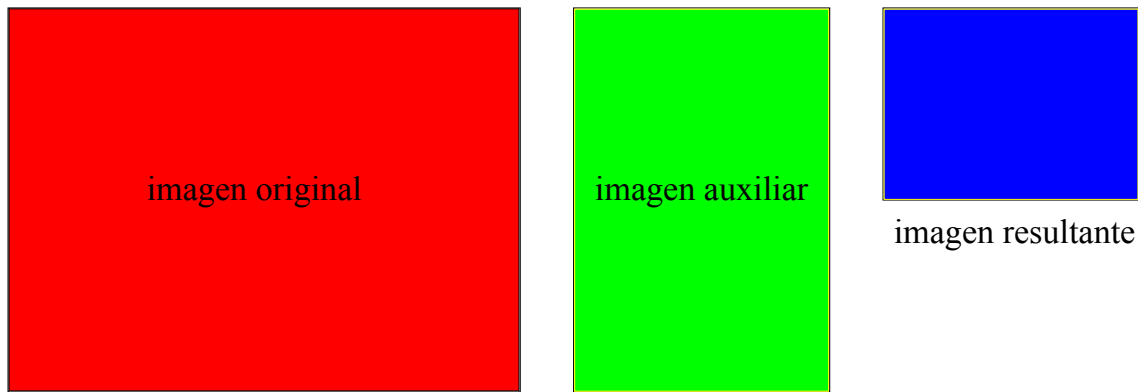


Figura 3.5: Relación entre las imágenes en consideradas en el filtro Gaussiano. Escala: 0,5.

hace es crear un nuevo núcleo Gaussiano en cada caso. En particular, para una imagen escalada de  $240 \times 180$  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 utilizados 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 requiere 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.

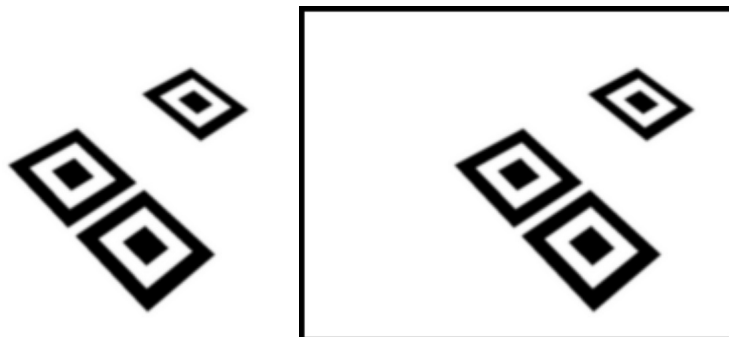


Figura 3.6: Imagen artificial del marcador trasladado y rotado, filtrada con el filtro Gaussiano. Izq.: Filtro Original. Der.: Filtro sin las condiciones de borde.

### 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 gradiente 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  $\tau$ , 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 concéntricos, 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 desempeño ni del tiempo de ejecución cuando tan sólo se enfoca al marcador. Sin embargo, si las imágenes capturadas cuentan con muchos segmentos (imágenes naturales genéricas), se ve que la detección de los mismos es menos precisa que la del algoritmo original, pero que los tiempos de procesamiento son notablemente inferiores.

### 3.6.4. Algoritmo 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. La imagen de prueba fue la de la figura 3.7; sépase que por cómo es el algoritmo, el contenido de la imagen es 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   | $480 \times 360$ | $480 \times 360$  |
| Escala                        | 0,5              | 0,5               |
| Tamaño de imagen de salida    | $240 \times 180$ | $240 \times 180$  |
| Segmentos detectados          | 36               | 36                |
| Tiempo medio de procesamiento | <b>36ms</b>      | <b>29ms</b>       |

Tabla 3.1: Comparación entre los tiempos de ejecución del filtro Gaussiano optimizado y el original. Ambos con precisión doble.

##### ■ Precisión simple (*float*)

|                               | Filtro original  | Filtro optimizado |
|-------------------------------|------------------|-------------------|
| Tamaño de imagen de entrada   | $480 \times 360$ | $480 \times 360$  |
| Escala                        | 0,5              | 0,5               |
| Tamaño de imagen de salida    | $240 \times 180$ | $240 \times 180$  |
| Segmentos detectados          | 36               | 36                |
| Tiempo medio de procesamiento | <b>28ms</b>      | <b>20ms</b>       |

Tabla 3.2: Comparación entre los tiempos de ejecución del filtro Gaussiano optimizado y el original. Ambos con precisión simple.

#### 3.6.5.2. Line Segment Detection

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



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 utilizada              | marcador sintético | marcador sintético   |
| Tamaño de imagen de entrada   | $480 \times 360$   | $480 \times 360$     |
| Escala                        | 0,5                | 0,5                  |
| Tamaño de imagen de salida    | $240 \times 180$   | $240 \times 180$     |
| Segmentos detectados          | 36                 | 36                   |
| Tiempo medio de procesamiento | <b>55,4ms</b>      | <b>48ms</b>          |

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 utilizada              | <i>zebras.png</i>  | <i>zebras.png</i>    |
| Tamaño de imagen de entrada   | $480 \times 360$   | $480 \times 360$     |
| Escala                        | 0,5                | 0,5                  |
| Tamaño de imagen de salida    | $240 \times 180$   | $240 \times 180$     |
| Segmentos detectados          | 251                | 179                  |
| Tiempo medio de procesamiento | <b>179,7ms</b>     | <b>94,4ms</b>        |

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 utilizada              | marcador sintético | marcador sintético   |
| Tamaño de imagen de entrada   | $480 \times 360$   | $480 \times 360$     |
| Escala                        | 0,5                | 0,5                  |
| Tamaño de imagen de salida    | $240 \times 180$   | $240 \times 180$     |
| Segmentos detectados          | 36                 | 36                   |
| Tiempo medio de procesamiento | <b>47,8ms</b>      | <b>38,8ms</b>        |

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 utilizada              | <i>zebras.png</i>  | <i>zebras.png</i>    |
| Tamaño de imagen de entrada   | $480 \times 360$   | $480 \times 360$     |
| Escala                        | 0,5                | 0,5                  |
| Tamaño de imagen de salida    | $240 \times 180$   | $240 \times 180$     |
| Segmentos detectados          | 252                | 182                  |
| Tiempo medio de procesamiento | <b>189,8ms</b>     | <b>90,8ms</b>        |

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 mediante 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 estimar la pose de una cámara, resulta necesario modelarla adecuadamente ya que no todas las cámaras son iguales. El modelo más comunmente utilizado es el denominado *pin-hole*. Para modelar completamente 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 [?]

#### 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 1.1.

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 arbitraria.

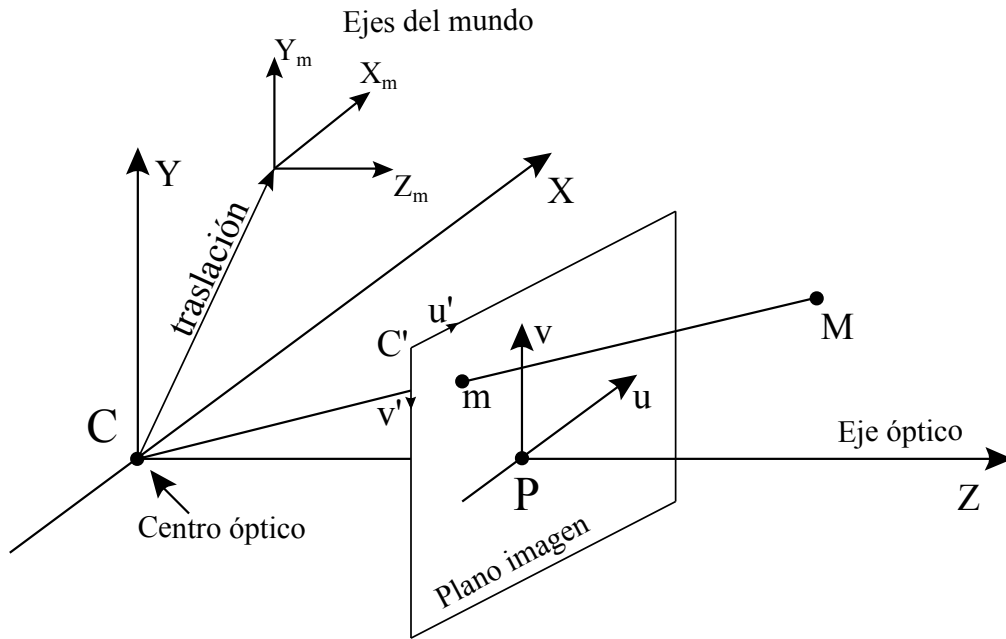


Figura 4.1: Modelo de cámara pin-hole.

- *Coordenadas de la cámara:* son las coordenadas que describen la posición del punto  $\mathbf{M}$  respecto de los ejes de la cámara  $(X, Y, Z)$ .
- *Coordenadas de la imagen:* son las coordenadas que describen la posición del punto 2D,  $\mathbf{m}$ , 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,  $\mathbf{m}$ , respecto del eje de coordenadas  $(u', v')$  situado en la esquina superior izquierda del plano imagen.

La transformación que lleva del punto  $\mathbf{M}$ , expresado respecto de las coordenadas del mundo, al punto  $\mathbf{m}$ , 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. Se le llama al conjunto de valores que definen esta transformación *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  $PY$  a la matriz proyección total, tal que:

$$\mathbf{m} = PY.M,$$

entonces:

$$PY = I.E$$

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



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 principal (**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”  $PY$  que dependa tanto de los parámetros intrínsecos de la cámara como de sus parámetros extrínsecos:

$$m = PY.M$$

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 [?].

Para determinar la forma de la matriz de proyección se estudia cómo se relacionan las coordenadas de **M** con las coordenadas 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 1.1 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  $(u'_0, v'_0, s'_0)$ :

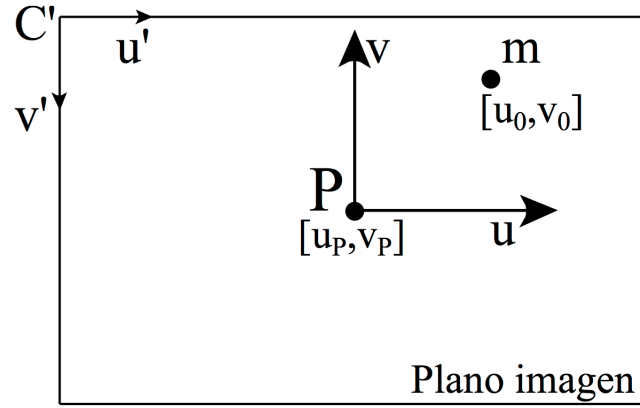


Figura 4.2: Relación entre el sistema de coordenadas de la imagen y el sistema de coordenadas normalizadas de la imagen.

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 imagen  $(u'_P, v'_P)$ . Las coordenadas de **m** dejan de ser expresadas en milímetros para ser expresadas en píxeles. Aparecen los factores de conversión  $d_u$  y  $d_v$ :

$$\begin{aligned} u'_0 &= d_u \cdot u_0 + u'_P \\ v'_0 &= d_v \cdot v_0 + v'_P \end{aligned}$$

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:

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

Nota: Este modelo no tiene en cuenta los efectos de distorsión de la lente.

- **Matriz de parámetros extrínsecos ( $E$ ):** de las coordenadas homogéneas del punto  $\mathbf{M}$  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 ( $PY$ ):** de las coordenadas homogéneas del punto  $\mathbf{M}$  expresadas respecto del sistema de coordenadas del mundo  $(X_{m0}, Y_{m0}, Z_{m0}, T_{m0})$ , a las coordenadas homogéneas del punto  $\mathbf{m}$  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:

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

### 4.3. Métodos para la calibración de cámara

Como se vió algunos párrafos atrás, el proceso mediante el cual se calculan los parámetros intrínsecos reales de una cámara es denominado “calibración de cámara”. Existen varios métodos para dicho proceso, entre los que se destacan el método de Zhang [?] y el método de Heikkilä y Silvén [?]. Como en este proyecto se utilizó una implementación basada en el primero de ellos (se hablará de dicha implementación un poco más adelante), este será explicado brevemente a continuación.

---

## CAPÍTULO 5

---

# POSIT: *POS* with *IT*erations

### 5.1. Introducción

En este capítulo se explica el algoritmo utilizado para el cálculo de la pose a partir de una imagen capturada por la cámara. Como lo dice el nombre de algoritmo se utiliza una técnica llamada *POS* (*Pose from Orthography and Scaling*), esta técnica consiste en aproximar la pose de la cámara a partir de la proyección *SOP* (*Scaled Orthographic Projection*). Se comienza el capítulo explicando en que consiste la proyección *SOP* y como se estima la pose a partir ella. Con esto como fundamento teórico se explican las diferentes variantes de POSIT y finalmente se explica la implementación utilizada en la aplicación.

### 5.2. POSIT clásico

La primera versión de POSIT presentada por *Daniel DeMenthon* y *Larry Davis* en [?] resuelve el problema de calcular la pose de la cámara dados 4 o más puntos detectados en la imagen y sus correspondientes en el mundo real, con la condición de que estos puntos no sean coplanares. Si bien no es la versión final que se utilizó vale la pena ser explicada ya que ayuda a sentar las bases de la implementación utilizada.

#### 5.2.1. Notacion y definición formal del problema de estimación de pose

En la figura 6.1 se puede ver un modelo de cámara pinhole, donde el centro es el punto  $O$ ,  $G$  es el plano imagen a una distancia focal  $f$  de  $O$ .  $O_x$  y  $O_y$  son los ejes que apuntan en las direcciones de las filas y las columnas del sensor de la cámara respectivamente.  $O_z$  es el eje que esta sobre el eje óptico de la cámara y apunta en sentido saliente. Los versores para estos ejes son  $\mathbf{i}$ ,  $\mathbf{j}$  y  $\mathbf{k}$ .

Se considera ahora un objeto con puntos característicos  $M_0, M_1, \dots, M_i, \dots, M_n$ , cuyo eje de coordenadas esta centrado en  $M_0$  y está compuesto por los versores  $(M_0u, M_0v, M_0w)$ . La geometría del objeto se asume conocida, por lo tanto las coordenadas de los puntos característicos del objeto en el eje de coordenadas del mismo son conocidas. Por ejemplo  $(U_i, V_i, W_i)$  son las coordenadas del punto  $M_i$  en el marco de referencia del objeto. Los puntos correspondientes a los puntos del objeto  $M_i$  en la imagen son conocidos y se identifican como  $m_i$ ,  $(x_i, y_i)$  son las coordenadas de este punto en la imagen. Las coordenadas de los puntos  $M_i$  en el eje de coordenadas de la cámara, identificadas como  $(X_i, Y_i, Z_i)$ , son desconocidas ya que no se conoce la pose del objeto respecto a la cámara.

Se busca entonces computar la matriz de rotación y el vector de traslación del objeto respecto a la cámara. La matriz de rotación  $\mathbf{R}$  del objeto, es la matriz cuyas filas son las coordenadas del los

versores  $i$ ,  $j$  y  $k$  del sistema de coordenadas de la cámara expresados en el sistema de coordenadas del objeto ( $u$ ,  $v$ ,  $w$ ).

La matriz  $\mathbf{R}$  queda:

$$R = \begin{pmatrix} i_u & i_v & i_w \\ j_u & j_v & j_w \\ k_u & k_v & k_w \end{pmatrix}$$

Para obtener la matriz de rotación solo es necesario obtener los versores  $\mathbf{i}$  y  $\mathbf{j}$ , el versor  $\mathbf{k}$  se obtiene de realizar el producto vectorial  $\mathbf{i} \times \mathbf{j}$ . El vector de traslación es el vector que va del centro del objeto  $M_0$  a el centro del sistema de coordenadas de la cámara  $O$ . Por lo tanto las coordenadas del vector de traslación son  $(X_0, Y_0, Z_0)$ . Si este punto  $M_0$  es uno de los puntos visibles en la imagen, entonces el vector  $\mathbf{T}$  esta alineado con el vector  $Om_0$  y es igual a  $(Z_0/f)Om_0$ . Como detalle a tener en cuenta, se observa que para calcular la traslación es necesario conocer el punto de referencia del objeto  $M_0$ , esta es la principal diferencia con la versión moderna de POSIT en la que para calcular la traslación no es necesario suponer nada acerca del punto de referencia del objeto.

Por lo tanto la pose queda determinada si se conocen  $\mathbf{i}$ ,  $\mathbf{j}$  y  $Z_0$ .

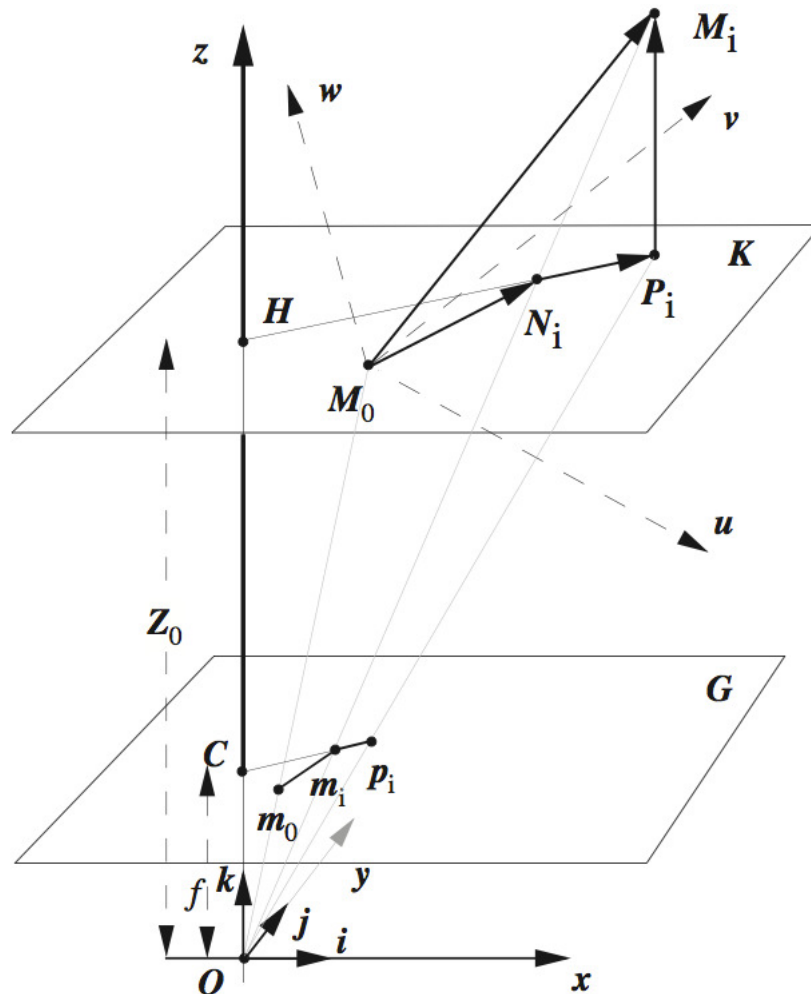


Figura 5.1: Proyección en perspectiva ( $m_i$ ) y SOP ( $p_i$ ) para un punto del modelo 3D  $M_i$  y un punto de referencia del modelo  $M_0$ . Fuente: [?] .

### 5.2.2. SOP: Scaled Ortographic Projection

La proyección ortogonal escalada(SOP) es una aproximación a la proyección perspectiva. En esta aproximación se supone que las profundidades  $Z_i$  de diferentes puntos  $M_i$  en el eje de coordenadas de la cámara no difieren mucho entre sí, y por lo tanto se asume que todos los puntos  $M_i$  tienen la misma profundidad que el punto  $M_0$ .

Para un punto  $M_i$  la proyección perspectiva sobre el plano imagen estaría dada por:

$$x_i = fX_i/Z_i, y_i = fY_i/Z_i,$$

, mientras que la proyección SOP esta dada por:

$$x'_i = fX_i/Z_0, y'_i = fY_i/Z_0.$$

De aquí en más las proyecciones SOP de los puntos  $M_i$  se identificaran como  $p_i$  mientras que las proyecciones persepectivas, que son los puntos que se detectan en la imagen, se identifican como  $m_i$ . Al término  $s = f/Z_0$  se lo conoce como el factor de escala de la SOP. Se puede ver que para el caso particular del punto  $M_0$  la proyeccion perspectiva  $p_0$  y la SOP  $m_0$  coinciden.

En la figura 6.1 se puede ver como se construye la SOP. Primero se realiza la proyección orotgona de todos los puntos  $M_i$  sobre  $K$ , el plano paralelo al plano imagen que pasa por el punto  $M_0$ . Las preyecciones de los puntos  $M_i$  sobre  $K$  se llaman  $P_i$ . El segundo paso consiste en hacer la proyección perspectiva de los puntos  $P_i$  sobre el plano imagen  $G$  para obtener finalmente los puntos  $p_i$ . En la figura también se puede ver que el tamaño del vector  $m_0p_i$  es  $s$  veces el tamaño de  $M_0P_i$ . Teniendo esto en cuenta se pueden expresar las coordenadas de  $p_i$  como:

$$\begin{aligned} x'_i &= fX_0/Z_0 + f(X_i - X_0)/Z_0 = x_0 + s(X_i - X_0) \\ y'_i &= y_0 + s(Y_i - Y_0) \end{aligned} \quad (5.1)$$

### 5.2.3. Ecuaciones para calcular la proyección perspectiva

Como se mencionó anteriormente la pose queda determinada si se conocen los vectores  $\mathbf{i}, \mathbf{j}$  y la coordenada  $Z_0$  del vector de traslación. Las ecuaciones que vinculan estas variables son:

$$M_0M_i \frac{f}{Z_0} \mathbf{i} = x_i(1 + \varepsilon_i) - x_0 \quad (5.2)$$

$$M_0M_i \frac{f}{Z_0} \mathbf{j} = y_i(1 + \varepsilon_i) - y_0 \quad (5.3)$$

donde  $\varepsilon_i$  se define como

$$\varepsilon_i = \frac{1}{Z_0} M_0M_i \mathbf{k} \quad (5.4)$$

Se puede ver que los terminos  $x_i(1 + \varepsilon_i)$  y  $y_i(1 + \varepsilon_i)$  son las coordeanadas  $(x'_i, y'_i)$  de la SOP. Primero se descompone el vector  $M_0M_i$  en la suma de  $M_0P_i$  y  $P_iM_i$ . De la figura 6.1 se puede ver que sentido del vector  $P_iM_i$  se según el versor  $\mathbf{k}$ . Al realizar el producto escalar con  $\mathbf{i}$  y  $\mathbf{j}$  se tiene que:

$$\begin{aligned} M_0M_i \mathbf{i} &= M_0P_i \mathbf{i} + P_iM_i \mathbf{i} = \frac{Z_0}{f} m_0p_i \mathbf{i} \\ M_0M_i \mathbf{j} &= M_0P_i \mathbf{j} + P_iM_i \mathbf{j} = \frac{Z_0}{f} m_0p_i \mathbf{j} \end{aligned} \quad (5.5)$$

Recordando que  $p_i$  es la SOP y que  $m_0$  es la proyección del punto de referencia del objeto, se tiene:

$$\begin{aligned} m_0 p_i \mathbf{i} &= x'_i - x_0 \\ m_0 p_i \mathbf{j} &= y'_i - y_0 \end{aligned} \quad (5.6)$$

Finalmente si se sustituye en 6.5 y se compara con 6.2 y 6.3 se puede ver que:

$$\begin{aligned} x'_i &= x_i(1 + \varepsilon_i) \\ y'_i &= y_i(1 + \varepsilon_i) \end{aligned} \quad (5.7)$$

#### 5.2.4. Algortímo

Las ecuaciones 6.2 y 6.3 se puede reescribir como:

$$M_0 M_i \mathbf{I} = x_i(1 + \varepsilon_i) - x_0 \quad (5.8)$$

$$M_0 M_i \mathbf{J} = y_i(1 + \varepsilon_i) - y_0 \quad (5.9)$$

### 5.3. POSIT moderno

### 5.4. SoftPOSIT

### 5.5. POSIT Coplanar

---

## CAPÍTULO 6

---

# Implementación

### 6.1. Introducción

En este capítulo se muestra la integración de los conocimientos adquiridos para poder llevar a cabo la realidad aumentada en una aplicación real. Si bien era de gran interés del proyecto la exploración de distintos métodos y algoritmos parecía importante poder poner en práctica todo lo desarrollado en un producto final que pudiera parecerse a un prototipo de aplicación comercial. La aplicación consta de distintas funcionalidades que se describen en este capítulo, tales como:

- (1) QR
- (2) Navegación por listas de cuadros
- (3) Servidor
- (4) Detección SIFT
- (5) Diferentes realidades aumentadas según la obra.

En las próximas secciones se describe más en detalle cada uno de estos puntos y su integración a la aplicación final.

### 6.2. Diagrama global de la aplicación

Para que sea más sencilla la comprensión de los bloques que componen la aplicación a continuación se muestra el *Storyboard* de la aplicación que es la interfaz de usuario y que sirve para visualizar bastante cada una de las clases que intervienen y cómo es el flujo de la aplicación.

IMAGEN DEL STORYBOARD COMPLETO

La aplicación cuenta con una pantalla de inicio con instrucciones en formato de audio al presionar el botón respectivo y una presentación sobre cómo es el recorrido. Se da la opción de elegir entonces una forma de recorrer el museo de manera automática, esto es, utilizando QRs para detectar la zona del museo en la que se está, procesamiento de imágenes (SIFT) para identificar el cuadro que se tiene enfrente y comunicación con un servidor que hace de procesador y comunicador de



resultados o una opción manual en la que el usuario es quien elige la zona del museo (autor) y el cuadro en el que está interesado.

Una vez que se está frente al cuadro existe una serie de interacciones habilitadas para el usuario, como ser: audioguía sobre el cuadro en particular, dibujo con lienzo libre para realizar tweets con la imagen dibujada, o la realidad aumentada vinculada a la obra que seleccionó (recorrido manual) o la que fue derivada luego del reconocimiento de imágenes (recorrido automático)

SEGUIR REDACTANDO ESTA SECCION

## 6.3. TableViewController

jjjjj

## 6.4. QR

### 6.4.1. QR. Una realidad

El uso de los identificadores QR (Quick Response), es cada vez más generalizado. Últimamente debido al incremento significativo del uso de *smart devices* el hecho de poder contar con cámara y poder de procesamiento hace que sea frecuente encontrar aplicaciones con el poder de reconocimiento de QRs. Comenzaron a utilizarse en la industria automovilística japonesa como una solución para el trazado en la línea de producción pero su campo de aplicación se ha diversificado y hoy en día se pueden encontrar también como identificatorios de entradas deportivas, tickets de avión, localización geográfica, vínculos a páginas web o en algunos casos también como tarjetas personales.

### 6.4.2. Qué son realmente los QRs?

Se puede decir que los QRs tienen muchos puntos en común con los códigos de barras pero con la ventaja de poder almacenar mucho más información debido a su bidimensionalidad. Existen distintos tipos de QRs, con distintas capacidades de almacenamiento que dependen de la versión, el tipo de datos almacenados y del tipo de corrección de errores. En su versión 40 con detección de errores de nivel L, se pueden almacenar alrededor de 4300 caracteres alfanuméricos o 7000 dígitos (frente a los 20-30 dígitos del código de barras) lo cual lo hace muy flexible para cualquier tipo de aplicación de identificación.

En la figura 7.1 se pueden ver las distintas partes que componen un QR como ser el bloque de control compuesto por las tres esquinas que dan información de la posición, alineamiento y sincronismo, así como también información de versión, formato, corrección de errores y datos. Fuera de toda esa información que podríamos denominar encabezado haciendo analogía con los paquetes de las redes de datos se encuentra la información a almacenar propiamente dicha que conforma el cuerpo del QR.

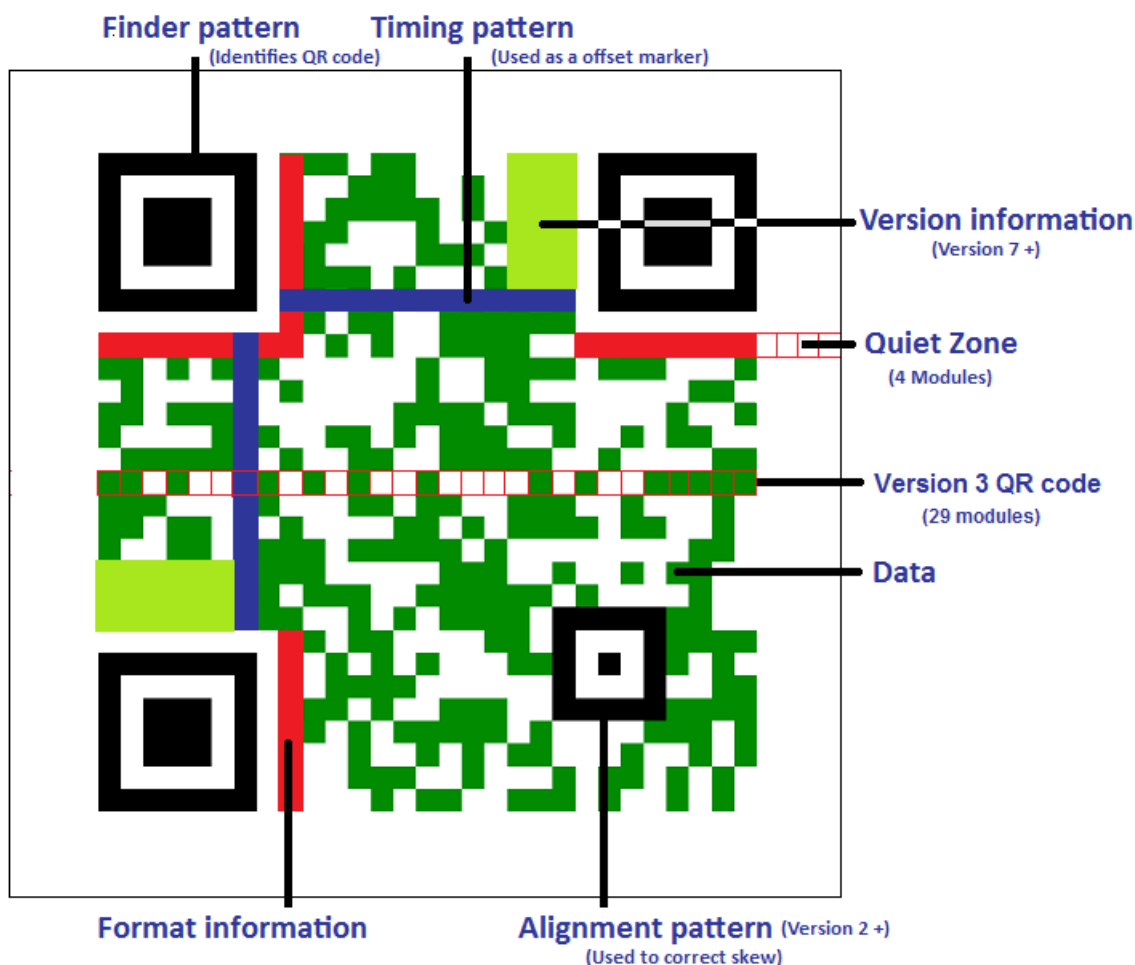


Figura 6.1: Las distintas componentes de un QR. Fuente [?].

### 6.4.3. Codificación y decodificación de QRs

Es fácil darse cuenta que la codificación resulta mucho más sencilla que la decodificación. Para la codificación es necesario comprender el protocolo, las distintas variantes y el tipo de información que se pretende almacenar. Sin embargo para la decodificación, además de tener que cumplir con lo anterior, es necesario contar con buenos sensores y ciertas condiciones de luminosidad y distancia que favorezcan a la cámara y se traduzcan en buenos resultados luego de la detección de errores. Si bien la plataforma es importante para lograr buenos resultados, dada una plataforma, existen variadas aplicaciones tanto para iOS como para Android que cuentan con performances bastante diferentes en función del algoritmo de procesamiento utilizado.

Debido a que el centro del proyecto de fin de carrera no fue la codificación y decodificación de QRs y que además ya existen distintas librerías que resuelven este problema se optó por investigar las distintas variantes e incorporar la más adecuada para la aplicación.

Dentro de todas las librerías que resuelven la decodificación se encuentran ZXing y ZBar como las más destacadas por su popularidad, simplicidad y buena documentación para la fácil implementación. ZXing, denominada así por "Zebra Crossing", es una librería open-source desarrollada en java y que tiene implementaciones que están adaptadas para otros lenguajes como C++, Objective C o JRuby entre otros.

Por su parte ZBar también tiene soporte sobre varios lenguajes y cuenta con un SDK interesante para desarrollar fácilmente aplicaciones que integren el lector de QR. Se trabajó sobre el código de

ejemplo que contiene la implementación de las clases principales para obtener un lector de QRs. Básicamente consta de una clase *ReaderSampleViewController* que hereda de *UIViewController* y que implementa un protocolo llamado *ZBarReaderDelegate*. Al presionarse el botón de detección se crea una instancia de la clase *ReaderSampleViewController* y se presenta la vista de cámara. Luego el protocolo se encarga de la captura y procesamiento del QR teniendo como resultado la información que tiene el QR en la variable denominada *ZBarReaderControllerResults*. Esta variable luego se mapea en una hash table con el contenido en formato *NSDictionary*. De esta manera se accede fácilmente al contenido en formato legible y es fácil de hacer una lógica de comparación y búsqueda en una base de datos.

#### 6.4.4. El QR en la aplicación

Para el caso particular de la aplicación se optó por tener un identificador QR para tres artistas elegidos del Museo Nacional de Artes Visuales (MNAV). Los mismos fueron Pedro Figari, Joaquín Torres García, Juan Manuel Blanes. De esta manera para el caso del recorrido del museo a través de la utilización con QRs es posible determinar la posición del usuario debido a imágenes QR debidamente ubicadas en cada zona. Esto sirve como localización y también sirve para lograr que el paso siguiente, que es la identificación de la obra que el usuario tiene enfrente, sea mediante una búsqueda en una base de datos discriminada por autor. Es decir, si el usuario no escanea el QR la búsqueda de la obra a identificar se hará en una base de datos global del museo, pero para el caso que el usuario sí decida escanear el QR entonces se cuenta con la posibilidad de realizar la búsqueda en una base de datos más reducida.

#### 6.4.5. Arte con QRs

La opción de usar los QRs de una manera distinta ha comenzado a ser notoria en los últimos tiempos. Hay quienes desafían a la información *cruda de 1s y 0s* incorporando imágenes y modificando colores y contornos en los QRs tradicionales para lograr un valor estético además del funcional. A continuación se muestran algunos ejemplos de tales casos en los que claramente se ve cómo puede lograrse el mismo resultado de información con el valor agregado de originalidad.

### 6.5. Servidor

Si bien el desarrollo de la aplicación es un prototipo de una aplicación comercial y para tal caso no se manejan muchas imágenes y otros datos y registros, para lograr escalabilidad se hace imprescindible contar con un servidor. Se pensó con el fin de almacenar toda aquella información relevante en cuanto a registro de obras (imagen, título y autor), descripciones de obras, audioguías, videos, modelos y animaciones para las realidades aumentadas asociadas y cualquier tipo de información que el museo quiera agregar y que por un tema de practicidad no se quiera almacenar dentro de la aplicación. En definitiva, almacenar toda esa información dentro de la aplicación quizá sea rentable para pocas obras, pero lo puede hacer inmanejable para un buen número de obras. Se pensó entonces en la instalación de un servidor que esté ubicado dentro del museo con el cual se tenga una conexión a través de una LAN de (54Mbps). Se aclara este punto pues, en caso de querer hacer un servidor remoto que tenga que ser accedido a través de internet, entonces todo es más lento, aunque funciona perfectamente.

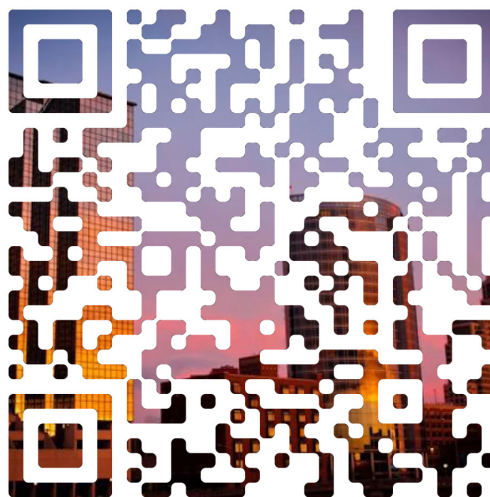


Figura 6.2: Ejemplo de un QR artista. Fuente [?].

### 6.5.1. Creando el servidor

Para la creación del servidor se buscó primeramente la alternativa de hacerlo sobre una máquina con sistema operativo con núcleo Linux, distribución Ubuntu. Luego también se buscó la posibilidad de tener el servidor corriendo sobre una plataforma Unix iOS. Para el segundo caso resultó incluso más sencilla que la primera dado que ya viene pensado por el sistema operativo el hecho de que funcione como servidor. A continuación se explica los pasos que se siguieron para implementar servidores en uno y otro sistema operativo.

#### 6.5.1.1. Servidor iOS

redactar pasos...

#### 6.5.1.2. Servidor LAMP

Se denota servidor LAMP por las siglas de Linux (Sistema Operativo), Apache (Servidor Web), MySQL (Gestor de base de datos), PHP/Perl/Python (lenguaje de programación).

Se instaló entonces el servidor Web Apache, que tiene dentro de sus principales ventajas el hecho de ser multiplataforma, gratis y de código abierto. Para eso desde terminal se debe hacer lo siguiente:

```
sudo apt-get install apache2
```

Con este comando se descarga el paquete *apache2* y se instala. Una vez finalizada la instalación de este paquete ya se cuenta con un servidor y se puede verificar ingresando desde la máquina donde se instaló el servidor abriendo el navegador e ingresando a *http://localhost* o equivalentemente a *http://127.0.0.1* y de esta manera aparece la página por defecto cuyo contenido está dado por el archivo */var/www/index.html*.

Luego de tener instalado el Apache se procede a instalar el php de la siguiente manera

```
sudo apt-get install php5
```

Para el caso particular de los intereses del servidor creado no fue necesario instalar MySQL. Entonces con esto, luego de reiniciar el servidor apache ya se tiene un servidor con intérprete php instalado. A partir de este momento todo se reduce a comprender bien el lenguaje php y poder

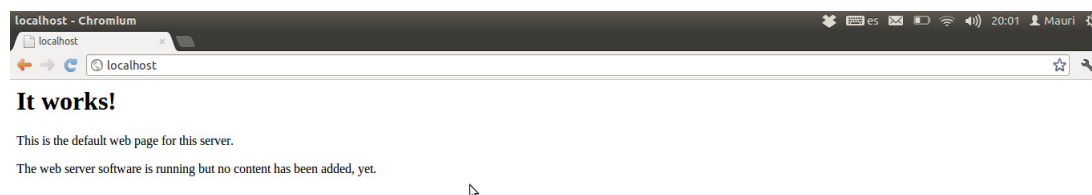


Figura 6.3: Impresión de pantalla al ingresar a <http://localhost>.

realizar pequeños módulos de programación que puedan tomar entradas, procesarlas y arrojar una salida.

### 6.5.2. Lenguaje php y principales scripts

Como fue dicho en la sección anterior para los intereses el servidor creado, lo fundamental es el hecho de poder recibir archivos o identificadores de los mismos, poder realizar un procesamiento en el servidor y devolver a la máquina cliente un archivo, mensaje o similar. Para esto se pasa a explicar algunos conceptos básicos de php.

El análogo a un Hello World en php es así:

```
<?php
echo"Hola Mundo";
?>
```

Esto se guarda en un archivo que con extensión .php en la ruta por defecto donde se alojan los php /var/www/holamundo.php. De esta manera al ingresar a la dirección <http://localhost/holamundo.php> se ve el print del texto.

como leer un input y hacer algo (ejemplo suma.php)

como hacer un accion.php ....

MOU SIGUE ESTO...

## 6.6. SIFT

## 6.7. Incorporación de la realidad aumentada a la aplicación

asdfasdfasdf

[?].

---

# Bibliografía

- [1] J. García Ocón. Autocalibración y sincronización de múltiples cámaras plz. 2007.
- [2] B. Furht. *The Handbook of Augmented Reality*. 2011.
- [3] C. Avellone and G. Capdehourat. Posicionamiento indoor con señales wifi. 2010.
- [4] Philip David, Daniel Dementhon, Ramani Duraiswami, and Hanan Samet. Simultaneous pose and correspondence determination using line features. pages 424–431, 2003.
- [5] Philip David, Daniel Dementhon, Ramani Duraiswami, and Hanan Samet. Softposit: Simultaneous pose and correspondence determination. pages 424–431, 2002.
- [6] Daniel F. DeMenthon and Larry S. Davis. Model-based object pose in 25 lines of code. *International Journal of Computer Vision*, 15:123–141, 1995.
- [7] R. Grompone von Gioi, J. Jakubowicz, J. M. Morel, and G. Randall. Lsd: A fast line segment detector with a false detection control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(4):722–732, April 2010.
- [8] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.
- [9] V. Lepetit and P. Fua. Monocular model-based 3d tracking of rigid objects: A survey. *Foundations and Trends in Computer Graphics and Vision*, 1(1):1–89, 2005.
- [10] Zhengyou Zhang. Flexible camera calibration by viewing a plane from unknown orientations. In *ICCV*, pages 666–673, 1999.
- [11] Janne 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.