Índice general

CAPÍTULO 1

Detección

Hola

1.1. Primitivas

1.1.1. Puntos y esquinas

1.1.1.1. Algortimos de puntos y esquinas

1.1.2. Líneas

1.1.2.1. Algortimos de líneas

1.1.3. Segmentos

1.1.3.1. Segmentos

1.1.4. Cilindros y esferas

1.1.4.1. Algortimos de cilindros y esferas

1.2. Detección sin primitivas markerless

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

1.3. Marcadores

Marcadores que se usan. Limitaciones

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

Se distinguen dos tipos de *fiduciales*. El primer tipo son los que se llaman puntos *fiduciales* por que proveen una correspondencia de puntos entre la escena y la imagen. El segundo tipo, *fiduciales planares*, se pueden obtener mediante la construcción en una geometría coplanar de una serie de

puntos fiduciales identificables como esquinas. Un único *fiducial planar* puede contener por si solo todas las seis restricciones espaciales necesarias para definir el marco de coordenadas.

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

1.3.1. Marcador QR

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

El marcador utilizado está basado en la estructura de detección incluida en los códigos *QR* y se muestra en la figura 1.1. Éste consiste en trés grupos idénticos de tres cuadrados concéntricos superpuestos de tal forma que los lados de cada uno de trés cuadrados son visualizables. A diferencia de los códigos *QR* la disposición de los grupos de cuadrados se distintos para evitar ambiguedades en la determinación de su posicionamiento espacial. Estas dos características son escenciales para la extracción de los *puntos fiduciales* de forma coherente, es decir, las correspondencias tienen que poder ser determinadas completamente bajo criterios razonables.

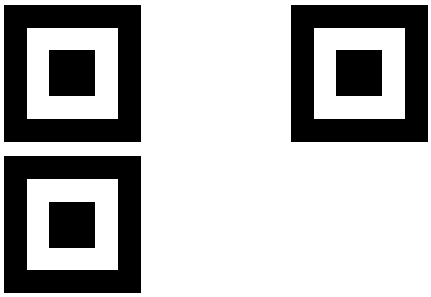

Figura 1.1: Marcador propuesto basado en la estructura de detección de códigos QR.

1.3.1.1. Diseño: estructura del marcador

En la presente subsección se presentan algunas definiciones de las estructuras formantes del marcador. Estas serán de utilidad para el diseño y formaran 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 serán los lados del cuadrilátero, el próximo elemento en la estructura del marcador.

Un *cuadrilátero* o `Ql` está determinado por cuatro segmentos conexos y distintos entre sí. La propiedad de conexión se relaja a la intersección de dos discos de un cierto radio en torno a los puntos de dos segmentos vecinos. El cuadrilátero tiene dos propiedades notables; el *centro* definido como el punto medio de sus vértices y el *perímetro* definido como la suma de sus cuatro lados. Los *vértices* de un cuadrilátero se determinan mediante la intersección, en un sentido amplio, de dos segmentos contiguos.

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

Un *grupo de cuadriláteros* o `QlSet`, se construye a partir de `M` cuadriláteros, de distinto tamaño, que comparten un mismo centro, con $\texttt{M} > 1$. A partir de dichos cuadriláteros se construye un lista ordenada $(\texttt{Ql[0]}, \texttt{Ql[1]}, \ldots, \texttt{Ql[M-1]})$ en donde el orden viene dado por el valor de perímetro de cada `Ql`. Se define el *centro del grupo de cuadriláteros* como el baricentro, o promedio ponderado, de los centros de cada `Ql` de la lista ordenada.

Finalmente el *marcador QR* estará constituido por `N` *grupos de cuadriláteros* dispuestos en una geometría particular. Esta geometría debe determinar un sistema de coordenadas, un origen y dos ejes. Se tendrá una lista ordenada $(\texttt{QlSet[0]}, \texttt{QlSet[1]}, \ldots, \texttt{QlSet[N-1]})$ en donde el orden se determinará mediante la geometría de los mismos.

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

1.3.1.2. Diseño

Un detalle del marcador se muestra en la figura 1.2 en donde se define el grupo `i` de cuadrados concentricos como el `QlSet[i]` (sets de cuadriláteros) y se definen los respectivos centros $\mathbf{c}_i$ para cada `QlSet[i]`. Se considera además un eje de coordenadas que queda definido los vectores normalizados.

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

Por otro lado la disposición de los `QlSet` es tal que la distancia indicada $\mathbf{d}_{01}$ definida como la norma del vector entre los centros $\mathbf{c}_1$ y $\mathbf{c}_0$ es significativamente mayor que la distancia $\mathbf{d}_{02}$ definida como la norma del vector entre los centros $\mathbf{c}_2$ y $\mathbf{c}_1$.

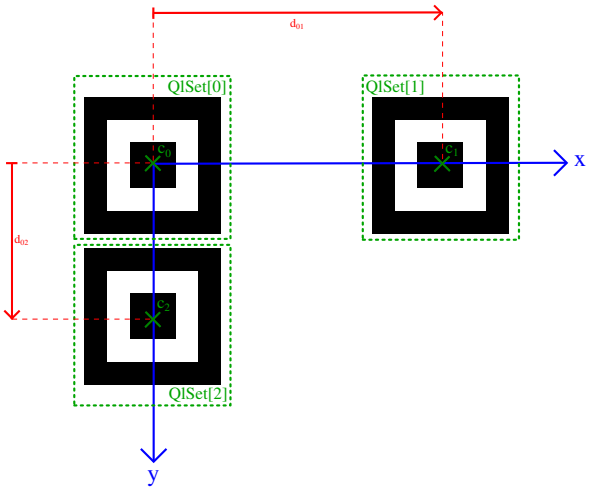

Figura 1.2: Detalle del marcador propuesto formando un sistema de coordenadas.

En base al sistema de coordenadas definido en la figura 1.2 se puede fijar un orden determinado para los *puntos fiduciales* que serán utilizados como correspondencias entre la imagen y la escena. Éstos se toman partiendo del cuadrado

1.3.1.3. Filtrado de segmentos

1.3.1.4. Determinación de correspondencias

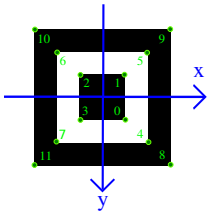

Figura 1.3: Detalle de un QlSet indicando el orden de los puntos basados en el eje de coordenadas definido previamente.

CAPÍTULO 2

LSD: “Line Segment Detection”

2.1. Introducción

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

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

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

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

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

2.2. *Line-support regions*

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

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

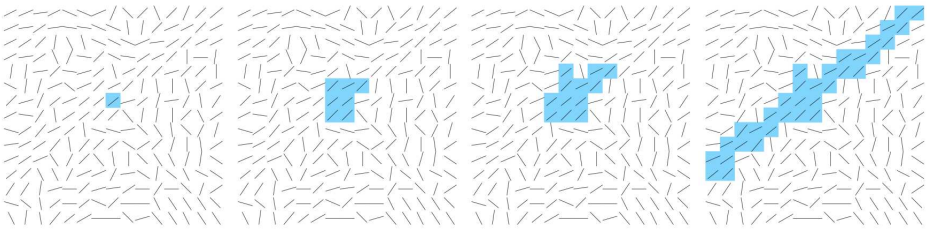

Figura 2.1: Proceso de crecimiento de una región. El ángulo asociado cada píxel de la imagen está representado por los pequeños segmentos y los píxeles coloreados representan la formación de la región. Fuente [7].

Los píxels agregados a una región son marcados de manera que no vuelvan a ser testeados. Para mejorar el desempeño del algoritmo, las regiones comienzan a evaluarse por los píxeles con gradientes de mayor amplitud ya que estos representan mejor los bordes.
Existen algunos casos puntuales en los que el proceso de búsqueda de *line-support regions* puede arrojar errores. Por ejemplo, cuando se tienen dos segmentos que se juntan y que son colineales a no ser por la tolerancia τ descripta anteriormente, se detectarán ambos segmentos como uno solo; ver figura 2.2. Este potencial problema es heredado del algoritmo de Burns, Hanson y Riseman.

Figura 2.2: Potencial problema heredado del algoritmo de Burns, Hanson y Riseman. Izq.: Imagen original. Ctro.: Segmento detectado. Der.: Segmentos que deberían haberse detectado. Fuente [7].

Sin embargo, LSD plantea un método para ahorraste este tipo de problemas. Durante el proceso de crecimiento de las regiones, también se realiza la aproximación rectangular a dicha región (paso (2) de los tres definidos anteriormente); y si menos del 50 % de los píxeles dentro del rectángulo corresponden a la *line-support region*, entonces lo que se tiene no es un segmento. Se detiene entonces el crecimiento de la región.

2.3. Aproximación de las regiones por rectángulos

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

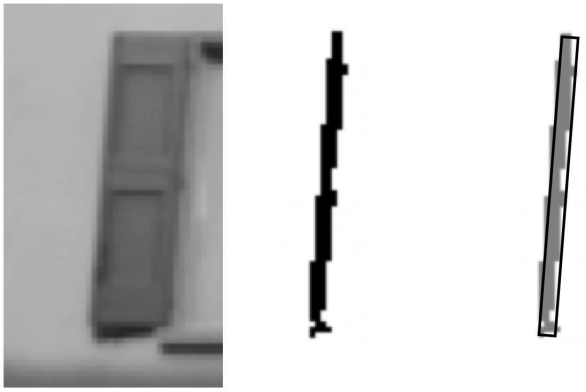

Figura 2.3: Búsqueda del segmento que mejor aproxime cada *line-support region*: aproximación de una región por un rectángulo. Izq.: Imagen original. Ctro.: Una de las regiones computadas. Der.: Aproximación rectangular que cubre el 99% de la masa de la región. Fuente [7].

2.4. Validación de segmentos

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

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

véase que el valor se logra multiplicando 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 empiricamente $\varepsilon = 1$ 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 el gradiente y el *level-line orientation field*. De esta menera, definido sobre ésta un segmento con cierta orientación, la probabilidad de que uno de sus píxeles

cuente con la misma orientación, a menos de la ya mencionada tolerancia τ; será:

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

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

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

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

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

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

2.5. Refinamiento de los candidatos

Por lo que se vió hasta el momento, la mejor aproximación rectangular a una *line-support region* es la que obtenga un valor NFA menor. Para los segmentos que no son validados, se prueban algunas variaciones a la aproximación original con el objetivo de disminuír su valor NFA y así entonces validarlos. Esta claro que este paso no es significativo para segmentos largos y bien definidos, ya que estos serán validados en la primera inspección; sin embargo, ayuda a detectar segmentos más pequeños y algo ruidosos.
Lo que se hace es probar distintos valores para la anchura del segmento y para sus posiciones laterales, ya que estas son los parámetros peor estimados en la aproximación rectangular, pero tienen un efecto muy grande a la hora de validar los segmentos. Es que un error de un píxel en el ancho de un segmento, agrega una gran cantidad de píxeles no alineados a este, y esto se ve reflejado en un valor mayor de NFA y puede llevar a una no detección.
Otro método para el refinamiento de los candidatos es la disminución de la tolerancia τ. Si los puntos dentro del rectángulo efectivamente corresponden a un segmento, aunque la tolerancia disminuya, se computará prácticamente misma cantidad de segmentos alineados, y con una probabilidad menor de ocurrencia ($\frac{\tau}{\pi}$), el valor NFA obtenido será menor. Los nuevos valores testeados de tolerancia son: $\frac{\tau}{2}$, $\frac{\tau}{4}$,$\frac{\tau}{8}$,$\frac{\tau}{16}$ y $\frac{\tau}{32}$. El valor NFA asociado al segmento será el menor de todos los calculados.

2.6. Optimización del algoritmo para tiempo real

2.7. Resutados

CAPÍTULO 3

Implementación

3.1. 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.

3.2. Diagrama global de la aplicación

bla bla laaa

3.3. TableViewController

jjjjj

3.4. QR

3.4.1. 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 automovolística 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.

3.4.2. 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 **??** 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.

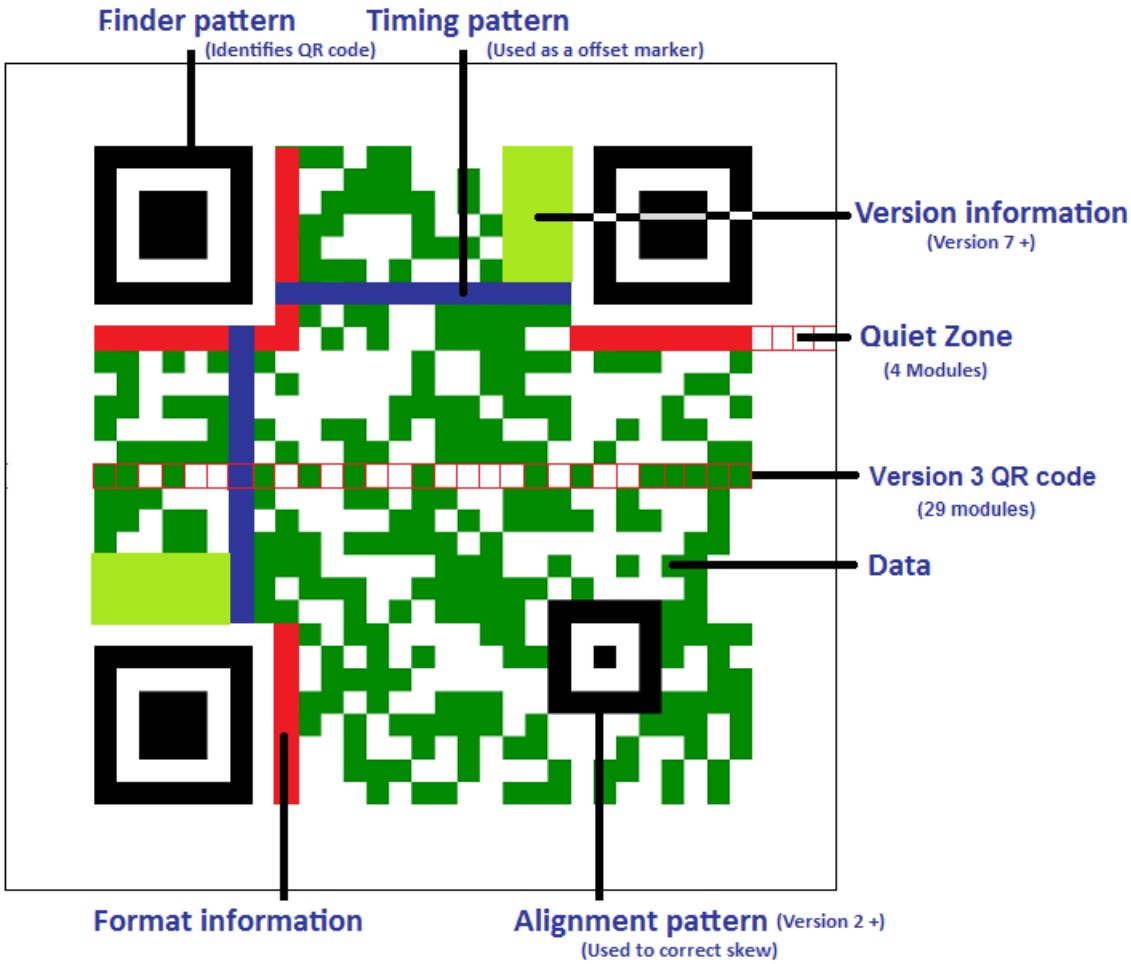

Figura 3.1: Las distintas componentes de un QR. Fuente [7].

3.4.3. 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++, Objetive 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 *Key* denominada *ZBarReaderControllerResults*. De esta manera se accede fácilmente al contenido

3.4.4. Arte con QRs

La opci´on de usar los QRs de una manera distinta

Figura 3.2: Ejemplo de un QR artista. Fuente [7].

3.5. Servidor

ssssssssssssss

3.6. SIFT

3.7. Incorporación de la realidad aumentada a la aplicación

asdfasdfasdf

[8].

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.