

---

---

# CAPÍTULO 1

---

## Casos de Uso

### 1.1. Introducción

En este capítulo se describen los distintos casos de uso que se implementaron con el fin de aplicar los algoritmos desarrollados en los capítulos anteriores. Se buscó generar distintos casos de uso que funcionaran como muestra de las funcionalidades que son posibles de realizar mediante la resolución de los algoritmos mencionados.

### 1.2. Caso de Uso 01

#### 1.2.1. Comentarios sobre el caso de uso

#### 1.2.2. Detalles constructivos

### 1.3. Caso de Uso 02

#### 1.3.1. Comentarios sobre el caso de uso

Este caso de uso básicamente busca desplegar un video en una superficie dada del mundo real. Esto puede ser de gran interés como complemento de contenido para un cuadro o cualquier obra si se piensa en aplicarlo para museos. Es posible por ejemplo, generar un video que sea reproducido dentro de los marcos del propio cuadro, en un extremo o en una superficie cualquiera que resulte interesante desde el punto de vista artístico. A continuación se explican brevemente algunos detalles para lograr la implementación de este caso de uso.

#### 1.3.2. Detalles constructivos

Para lograr lo propuesto para este caso de uso se implementó un proyecto que proyecta el video en uno de los cuadrados del marcador. De esta manera, de toda la lógica de estimación de pose, solamente se hace uso de la detección y filtrado. En particular no se hace uso de los resultados del posit. Teniendo entonces detectados los cuatro puntos en los que se quiere reproducir el video parecería que el problema está resuelto. Sin embargo, xcode no permite posicionar en forma directa una vista de video en cualquier conjunto de cuatro puntos.

Si simplemente se quiere reproducir un video, y no se quiere procesar el contenido, lo más cómodo para hacerlo es utilizar la clase *MPMoviePlayerController* que hereda de *NSObject*. Una alternativa

similar es haciendo uso de la clase *MPMoviePlayerViewController* que hereda de *UIViewController* y tiene como única propiedad una del tipo *MPMoviePlayerController*.

*MPMoviePlayerController* tiene un atributo *view* del tipo *UIView* que es la vista y es este atributo el que se quiere posicionar en los cuatro puntos detectados por el filtro. Un atributo del tipo *UIView* tiene un atributo *frame* que es del tipo *CGRect*

```
theMovie.view.frame = CGRectMake(0, 0, 60, 60);
```

En el código anterior *theMovie* es del tipo *MPMoviePlayerController*. De esta manera, se tiene el inconveniente de que en principio cualquier video parecería que solamente puede ser reproducido sobre rectángulos y no en cualquier polígono de cuatro puntos por ejemplo. Sin embargo algo que sí se puede hacer a las instancias de la clase *UIView* es una transformación afin o incluso, de manera más genérica, una homografía.

### 1.3.3. *CGAffineTransform* y *CATransform3D*

La clase *UIView* tiene una propiedad llamada *transform* que es del tipo *CGAffineTransform*. También tiene una propiedad llamada *layer*

### 1.3.4. Resolución de Homografía

A continuación se hace el desarrollo de la resolución del sistema de ecuaciones que se tuvo que resolver para hallar los parámetros de la homografía que transforma una imagen de referencia en la imagen que se tiene en cada momento como resultado de la captura de la cámara. Se asume entonces que se conocen los puntos de referencia y los puntos de referencia transformados (los detectados luego del filtrado de segmentos) y lo que se quiere averiguar es la matriz  $h$  que logra dicha transformación. Esta homografía 2D-2D se puede expresar en forma matricial, en coordenadas homogéneas de la siguiente manera:

$$\begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} i \\ j \\ k \end{pmatrix}$$

donde la matriz  $h$  representa la transformación, el vector  $(x, y, z)^t$  representa los puntos de referencia a ser transformados y el vector  $(i, j, k)^t$  respresenta los puntos detectados cuadro a cuadro como las esquinas del marcador.

Asumiendo un valor unitario para las coordenadas  $z$  y  $k$  la resolución del sistema se simplifica mucho y no se pierde generalidad. Imponiendo esto entonces, el sistema anterior se puede expresar de la siguiente forma:

$$xh_{11} + yh_{12} + h_{13} = i \quad (1.1)$$

$$xh_{21} + yh_{22} + h_{23} = j \quad (1.2)$$

$$xh_{31} + yh_{32} + h_{33} = 1 \quad (1.3)$$

Multiplicando la ecuación 1.3.4 por  $i$  e igualándola a la ecuación 1.1 se obtiene lo siguiente:

$$xh_{11} + yh_{12} + h_{13} = ixh_{31} + iyh_{32} + ih_{33} \quad (1.4)$$

o lo que es lo mismo:

$$xh_{11} + yh_{12} + h_{13} - ixh_{31} - iyh_{32} - ih_{33} = 0 \quad (1.5)$$

Procediendo de manera análoga y multiplicando la ecuación (1.3.4) por  $j$  e igualándola a la ecuación 1.2 se obtiene lo siguiente:

$$xh_{21} + yh_{22} + h_{23} = jxh_{31} + jyh_{32} + jh_{33} \quad (1.6)$$

o lo que es lo mismo:

$$xh_{21} + yh_{22} + h_{23} - jxh_{31} - jyh_{32} - jh_{33} = 0 \quad (1.7)$$

Las ecuaciones 1.3.4 y 1.3.4 se pueden expresar en forma matricial, de la siguiente manera:

$$\begin{pmatrix} x & y & 1 & 0 & 0 & 0 & -ix & -iy & -i \\ 0 & 0 & 0 & x & y & 1 & -jx & -jy & -j \end{pmatrix} \begin{pmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

## 1.4. Caso de Uso 03

### 1.4.1. Comentarios sobre el caso de uso

### 1.4.2. Detalles constructivos

## 1.5. Caso de Uso 04

### 1.5.1. Comentarios sobre el caso de uso

### 1.5.2. Detalles constructivos

[?].