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 superfice 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.
PONER FOTO MOSTRANDO EL CASO DE USO CON VIDEO PROYECTADO.

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*. Las primeras letras de esta clase (*CG*) refieren a la API **Core Graphics** utilizada ampliamente como herramienta para resolver *rendering* y cualquier tipo de transformación en 2D.
La clase *UIView* también tiene una propiedad llamada *layer* que es del tipo *CALayer* y que permite realizar transformaciones del tipo *CATransform3D*. Las primeras letras de estas dos clases (*CA*) refieren a la API **Core Animation** que es utilizada para generar animaciones y transformaciones sobre objetos 3D solamente indicando un punto inicial y final para el objeto (también es posible agregar efectos para la transición). En definitiva para resolver el problema del caso de uso existen a priori dos alternativas posibles: *CGAffineTransform* y *CATransform3D*.
Se pueden generar fácilmente instancias transformaciones afines invocando la siguiente función:

```
CGAffineTransform CGAffineTransformMake (
   CGFloat a,
   CGFloat b,
   CGFloat c,
   CGFloat d,
   CGFloat tx,
   CGFloat ty
);
```

que toma 6 *CGFloats* y crea una *CGAffineTransform*, donde cada uno de los valores anteriores se corresponde con los elementos de una matriz transformación afín de la siguiente manera:

$$\begin{pmatrix} a & b & 0 \\ c & d & 0 \\ t_x & t_y & 1 \end{pmatrix}$$

Así entonces, de los 9 valores de la matriz, 2 de ellos son nulos por tratarse de una transformación afín y otro de ellos es unitario como valor de escala. Resolviendo el sistema como se muestra en la sección 1.3.4 y obteniendo los restantes 6 valores, se le puede asignar transformaciones a la propiedad *transform* y realizar la trasnformación deseada. Este método tuvo como inconveniente el hecho de que .....
EXPLICAR POR QUE NO FUNCIONÓ

Por su parte también es sencillo generar instancias de transformaciones 3D debido a que existe el tipo de dato definido para generar la matriz *CATransform3D* como:

```
struct CATransform3D
{
CGFloat m11, m12, m13, m14;
CGFloat m21, m22, m23, m24;
CGFloat m31, m32, m33, m34;
CGFloat m41, m42, m43, m44;
};
typedef struct CATransform3D CATransform3D;
```

donde m_{ij} corresponde al elemento de la matriz ubicado en la fila i columna j. Así entonces también es posible, conociendo los valores de la homografía, completar los elementos de esta matriz 4x4 y asignársela a la propiedad *layer* de la *UIView*. Esta opción de generar una transformación 3D permite incluir transformaciones más generales que una homografía o una transformación afín. Si lo que se busca es que esta matriz represente una homografía (2D), es necesario entonces que la coordenada z sea nula, es decir

$$\begin{pmatrix} m_{11} & m_{12} & 0 & m_{14} \\ m_{21} & m_{22} & 0 & m_{24} \\ 0 & 0 & 1 & 0 \\ m_{41} & m_{42} & 0 & m_{44} \end{pmatrix}$$

donde a su vez m_{44} se asume de valor unitario por ser un factor de escala. De la misma manera que para la transformación afín, resolviendo la homografía como se ve en la sección 1.3.4 se obtienen los 8 valores restantes de la matriz y es posible asignarle una homografía a un objeto *UIView* para resolver el problema presente.

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_{3x3} representa la transformación homográfica, 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:

$$xh_{11} + yh_{12} + h_{13} = i \tag{1.1}$$

$$xh_{21} + yh_{22} + h_{23} = j \tag{1.2}$$

$$xh_{31} + yh_{32} + h_{33} = 1 \tag{1.3}$$

Multiplicando la ecuación (1.3) por i e igualándola a la ecuación (1.1) se obtiene lo siguiente:

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

o lo que es lo mismo:

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

Procediendo de manera análoga y multiplicando la ecuación (1.3) por j e igualándola a la ecuación (1.2) se obtiene lo siguiente:

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

o lo que es lo mismo:

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

Las ecuaciones (1.3) y (1.7) se pueden expresar en forma matricial, de la siguiente manera:

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

Teniendo entonces 4 parejas de puntos referencia y puntos transformados y asumiendo h_{33} de valor unitario se tiene entonces 8 ecuaciones y 8 incógnitas, lo que lo vuelve un sistema compatible determinado que se puede expresar de la siguiente manera:

$$\begin{pmatrix} x_0 & y_0 & 1 & 0 & 0 & 0 & -i_0x_0 & -i_0y_0 \\ 0 & 0 & 0 & x_0 & y_0 & 1 & -j_0x_0 & -j_0y_0 \\ x_1 & y_1 & 1 & 0 & 0 & 0 & -i_1x_1 & -i_1y_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -j_1x_1 & -j_1y_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -i_2x_2 & -i_2y_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -j_2x_2 & -j_2y_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -i_3x_3 & -i_3y_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -j_3x_3 & -j_3y_3 \end{pmatrix} \begin{pmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{pmatrix} = \begin{pmatrix} i_0 \\ j_0 \\ i_1 \\ j_1 \\ i_2 \\ j_2 \\ i_3 \\ j_3 \end{pmatrix}$$

Así entonces, lo que se hace para resolver la homografía es cuadro a cuadro tener detectados los puntos en los que se quiere presentar la vista del video que se corresponden con cuatro puntos detectados por el filtro y tener las correspondencias con el marcador real, se posiciona la vista en la posición de referencia y se le aplica la homografía hallada que vincula la posición referencia con los puntos detectados.

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

[**?**].