Visión Artificial

Entrega de Junio. 2020.

José Miguel Sánchez Almagro
FACULTAD DE INFORMÁTICA Universidad de Murcia.

Ejercicio RECTIF

En este ejercicio se trata la transformación de planos en dos dimensiones. El objetivo del mismo es, dada una fotografía que tiene una superficie plana y elementos de los que sabemos su longitud en el mundo, obtener la distancia o longitud de otros elementos que se encuentren en el mismo plano.

Se utilizarán dos fotografías, una de ellas es un escritorio que tiene un carnet de conducir, una regla, dos monedas, y un muñeco de Bart Simpson. En esta foto, se pide medir la distancia entre ambas monedas. Como vemos en el vídeo (enlace al final del ejercicio), la distancia obtenida cuando se mide la regla es buena, aunque no precisa, obteniendo 4’2 centímetros de 4.

La otra fotografía es una instantánea de un partido de fútbol. El objetivo, medir la distancia entre el jugador que ha realizado el disparo y la portería.

En primer lugar, se debe rectificar el plano de la imagen original. Para ello, obtenemos la homografía de la imagen, indicando cuatro puntos que formen el rectángulo de un objeto de dimensiones conocidas, en el caso de la primera imagen es el carnet, y en la segunda un rectángulo del terreno de juego del que conocemos sus dimensiones.

Con esa homografía somos capaces de reconstruir la imagen desde una perspectiva superior, con la que recuperamos la forma original de los objetos que se encuentran en el plano. Cuanto más nos alejemos del objeto que hemos seleccionado, más se distorsiona la imagen y peor será tanto la reconstrucción como la medición final.

En el script se ha utilizado el medidor del script *medidor.py*. Una vez que vemos la imagen rectificada se puede seleccionar en ella dos puntos, obteniendo la distancia en centímetros entre los mismos.

El script se llama pasando como parámetro un archivo .txt que contiene lo siguiente:

- El nombre del archivo que contiene la imagen.
- Los puntos de la imagen original que corresponden a los extremos del carné o del rectángulo.
- La nueva ubicación de esos puntos en la imagen reconstruida.
- La proporción píxel/milímetro.

Esta última es utilizada para medir las distancias. Una vez que sabemos el número de píxeles que hay entre dos puntos, debemos saber a que distancia corresponde en el mundo, y para eso utilizamos las medidas del objeto en el mundo real. Sabemos que el carné de conducir mide 8’5 centímetros de largo y 5’4 alto. El rectángulo seleccionado del campo de fútbol, 11x5’5 metros. Si la correspondencia píxel por milímetro es 2 a 1, significa que por cada dos píxeles estamos representando un milímetro del mundo real. Por supuesto, de manera aproximada. Con esos datos, si sabemos que entre el punto A y B hay 50 píxeles, tenemos que la distancia en el mundo real serían 50/2 milímetros, que diviendo por 10 son 2’5 centímetros. **Asumiendo que los puntos entre los que se mide la distancia se encuentran en el mismo plano que el rectángulo**.

El código no tiene nada relevante que no se haya explicado. En primer lugar, se intenta leer el archivo que contiene la información. En caso de que no se haya introducido ninguno o que éste no contenga la información necesaria el script parará mostrando un mensaje de error. En caso contrario, se leen todos los datos y se realiza la rectificación de la imagen a partir de los puntos proporcionados en el archivo dado como entrada. Una vez reconstruida la imagen, se produce el ya conocido bucle *autoStream()* que se encargará de imprimir la imagen reconstruida y los puntos que va seleccionando el usuario, así como la distancia entre los mismos.

En caso de haber dispuesto de más tiempo, se podrían haber realizado ciertas mejoras, como por ejemplo, mover la imagen reconstruida en la ventana en la que se muestra, para la mejor visualización del usuario.

Enlace del vídeo de demostración: https://youtu.be/L_sfF0aFsPU

Ejercicio AR

En este ejercicio se utilizan técnicas de realidad aumentada (RA de aquí en adelante) para proyectas imágenes sobre el vídeo del mundo real. La mayoría de técnicas de RA se basan en el uso de marcadores o patrones para proyectar las imágenes. En este caso, se utiliza la detección de un polígono de seis lados. Si se encuentra alguno, se proyectan las imágenes.

Para la detección del polígono se utiliza la función *polygons* proporcionada en un Notebook de la asignatura, que a su vez utiliza la función *redu* del repositorio de la asignatura. Otra función que se utiliza es *extractContours*, para detectar siluetas oscuras.

La imagen del polígono utilizada es la siguiente:

[IMAGEN]

Cuando se ha realizado una detección, debemos mostrar las imágenes en función de la posición el tamaño del marcador encontrado. De esta manera, el usuario podrá moverse alrededor del marcador observando las imágenes. Para representar imágenes se utiliza la función *applyImg()*, que dados cuatro puntos relativos (respecto al marcador), se transforman en los cuatro puntos a partir de los cuales se representará la imagen en la escena. Estos cuatro puntos se obtienen con la función *htrans()*, también del repositorio de la asignatura. Esos puntos obtenidos se relacionan con los puntos de las esquinas de la imagen (no los puntos en los que se colocará la imagen en el mundo) para obtener la homografía y realizar una rectificación, como en el ejercicio anterior, y poder así representar la imagen. El siguiente fragmento de código muestra la función:

```
def applyImg(src, M, world, img, frame):
    # calculamos dónde se proyectarán en la imagen esas esquinas
    # usamos la matriz de cámara estimada
    dst = htrans(M, world)

    # calculamos la transformación
    # igual que findHomography pero solo con 4 correspondencias
    H = cv.getPerspectiveTransform(src.astype(np.float32), dst.astype(np.float32))

    # la aplicamos encima de la imagen de cámara
    cv.warpPerspective(img,H,size,frame,0,cv.BORDER_TRANSPARENT)
```

Tal y como he dicho arriba, en este ejercicio se muestran cubos. Para formar un cubo se utiliza la misma imagen (color sólido) repetido 6 veces en 6 posiciones distintas. Esto se realiza en la función *drawBox()*, pero antes, hablaré sobre la representación de los cubos.

Para el manejo de los mismos se ha creado esta clase Box, que imita a las estructuras *struct* de otros lenguajes como C:

```
class Box:
    def __init__(self, x, z, y, size):
        self.x = x
        self.z = z
        self.y = y
        self.size = size
        self.xMaxSpeed = 0
        self.zMaxSpeed = 0
        self.yMaxSpeed = 0
        self.xSpeed = 0
        self.zSpeed = 0
        self.ySpeed = 0
        self.objective = [0,0,0]
        self.originalDistance = 0
```

En esta estructura se almacenan los puntos centrales del cubo (x, z, y), el tamaño del mismo, la velocidad máxima a la que podrá moverse en cada coordenada, la velocidad de movimiento actual, el punto destino en el caso de que se vaya a mover, y la distancia original respecto a ese punto. El uso de todo esto lo veremos más adelante.

Retomando la función *drawBox()*, sus argumentos son la imagen que hará de lado del cubo, la estructura Box, y M y frame, necesarios para la representación como hemos visto anteriormente. En esta función se calculan las coordenadas de cada imagen del cubo, en función de los atributos x, y, z y size de la estructura Box. No merece la pena entrar en más detalle sobre cómo se realiza el cálculo de las coordenadas de los puntos que utilizaremos para representar cada imagen. Una vez se han calculado los puntos, estos se pasan a la función *applyImg()* junto a la imagen para representar cada lado.

Una vez tenemos una función que dibuja los cubos, tenemos que moverlos. En un primer momento se quiso imitar el movimiento de los electrones de un átomo o de los planetas en el sistema solar, pero la falta de tiempo impidió tales objetivos. Tenemos tres cubos de color azul que se mueven alrededor de uno rojo que se mantiene estático. El movimiento de los mismos es arcaico. Se le marca un punto objetivo de destino, se realiza una resta entre dicho objetivo y la posición actual, y se mueve hacia dicho punto. Para obtener una sensación de curva y movimiento circular, se va aumentando o disminuyendo la velocidad de movimiento del cubo en función de cada coordenada. Para esto se utilizan un *distanceRatio* y un *closeRatio*. Cuanta mayor sea la distancia entre la posición actual y el objetivo mayor será el ratio de distancia (valor entre 0 y 1), u viciversa con el ratio de cercanía. Como hemos visto anteriormente, el cubo almacena sus coordenadas actuales, las de su punto de destino, y la distancia original cuando se estableció el destino. Esto necesario para calcular los ratios. Un último detalle a comentar, el ratio de distancia se *nerfea* reduciendo su valor, para un mejor movimiento circular.

```
distanceRatio = distance / exterior.originalDistance
closeRatio = 1 - distanceRatio
distanceRatio *= 0.9
```

Hay tres funciones que regulan el movimiento, *horizontalMovement()*, *leftDiagonalMovement()* y *rightDiagonalMovement()*. Los nombres hablan por si solos. Las tres funciones tienen la misma estructura, y su funcionamiento es el mismo. Para aplicar el

movimiento utilizan una cuarta función llamada *applyMovement()*. También se utiliza un *threshold* para cambiar el objetivo una vez que se ha llegado al destino con el margen de dicho atributo. El resto de detalles no merece la pena comentarlos.

Este código de movimiento es bastante arcaico, con más tiempo podría haberse mejorado, obteniendo movimientos más precisos que facilitaran la creación de elipses, por ejemplo.

Además, el diseño de los cubos y la escena en general es también mejorable.

Enlace del vídeo de demostración: https://youtu.be/5hbhmkB2MNY

Conclusiones

La asignatura gira entorno al tratamiento de imágenes y todas sus posibilidades. Hemos aprendido desde parámetros de cámaras, como la distancia focal, el ángulo FOV, etc., características de las mismas y tipos de fotos, como la distorsión de los bordes que no se produce con las cámaras actuales.

Hemos estudiado el tratamiento del color, y como este se utiliza para crear umbrales, y poder así encontrar patrones, discriminar escenas, o simplemente representar histogramas.

También hemos aplicado transformaciones a imágenes de todo tipo. Un notebook extenso sobre esto ha sido el de transformación de planos en 2 dimensiones.

Todo esto, para poder ser aplicado en algoritmos y poder crear técnicas a partir de otras, como ocurría en el caso de la realidad aumentada, que se basa en la transformación de planos en dos dimensiones para proyectar imágenes sobre una escena.

Hemos también aprendido a reconocer imágenes a partir de puntos clave, patrones, o de colores. Siendo esto algo muy útil para aplicaciones en el mundo real.

Hemos trabajado con el famoso efecto chroma, ampliamente utilizado en televisión y cine, viendo en profundidad como se produce. Su estudio me ha parecido personalmente de los más interesante de la asignatura, y su funcionamiento es incluso más sencillo de lo que imaginaba antes de estudiarlo.

Todo esto, aplicado de una forma práctica que permite poner en práctica todos los conocimientos y no dedicar demasiado tiempo a conceptos teóricos o matemáticos. Hemos usado el lenguaje Python, muy cómodo para tareas de programación y que no había sido utilizado en toda la carrera, por lo que gracias a esta asignatura hemos salido con cierto bagaje en el uso del mismo.

Hemos utilizado múltiples librerías, pero la más importante y sobre la que gira la asignatura es OpenCV. Proporciona numerosas y útiles herramientas para todo tipo de escenarios que se puedan plantear. Dicha librería tiene un amplio uso por la comunidad, y múltiples aplicaciones tal y como se puede observar con una rápida búsqueda en Youtube. Gracias a su uso en esta asignatura y haber visto lo interesante que es, no descarto utilizarla en algún proyecto personal que me pueda interesar.

Concluyendo, se aprenden muchos conceptos matemáticos y teóricos sobre el tratamiento de imágenes, de color o incluso de cámaras, pero además resulta muy útil la visión práctica de la asignatura, y hace que hayamos acabado disfrutando de los scripts que hemos ido realizando.