Prácticas Visión Artificial

Ejercicio FOV

Utilizaré la cámara de mi smartphone (Pocophone F1) para la realización de este ejercicio.

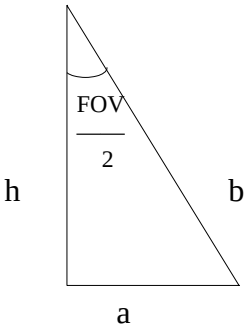
Podemos obtener el parámetro f de la cámara (Focal Length) con la siguiente fórmula *u=f(x/z)*. La imagen a usar será una realizada a un termo y tiene una resolución de **3024x4032** píxeles. La imagen es la siguiente:

La distancia entre la cámara y el termo son 29 centímetros (parámetro z), el tamaño del termo son 22 centímetros (parámetro x), y el número de píxeles que ocupa en la imagen es 3140-920 = 2220 px. (medido utilizando la altura/el largo de la botella).

Obtenemos por tanto el parámetro f de la cámara, necesario para saber el FOV. f = u / (x/z) = 2220 / (22/29) = **2926 px.**

Una vez tenemos la f, obtener el FOV es directo. FOV = 2 * arc tan ((w/2) / f) = 2 * arc tan ((4032/2) / 2926) = **69º**.

Para poder resolver la cuestión de a qué altura colocar la cámara para ver toda la cancha de baloncesto seguiremos este esquema:



Conocemos el largo de una pista de baloncesto cualquiera, 28 metros, y el ángulo FOV de la cámara, 69º. Podemos conocer h de dos maneras:

- $h = \sqrt{b^2-a^2}$
- h = a * arc tg (FOV/2)

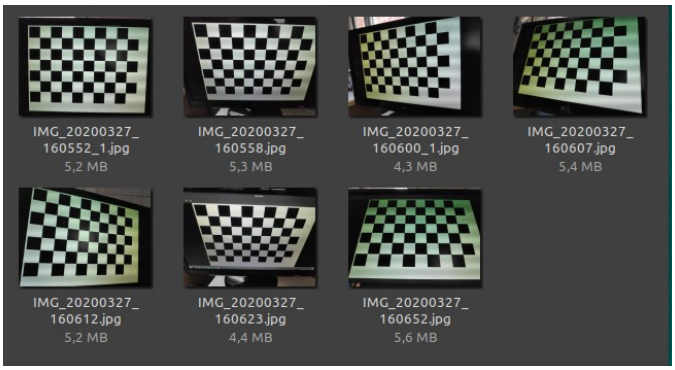
Utilizaremos la segunda, ya que contamos con todos los datos:

h = (28/2) * arc tg 34’5 = 14 * (1 / tg 34’5) = 20’37 metros.

La altura a la que debe estar la cámara del suelo para grabar toda la pista son 20’37 metros.

Para saber la altura de la pelota en cada momento cuando se está grabando un partido de baloncesto donde hemos colocado la cámara simplemente tenemos que resolver la misma operación que antes con el triángulo. La diferencia es que ahora, *a* serán los píxeles que ocupa el diámetro de la pelota.

Se han realizado las siguientes imágenes de la imagen del tablero de ajedrez (se adjuntan junto a este documento en la carpeta “chessboard”, al igual que la imagen del termo):

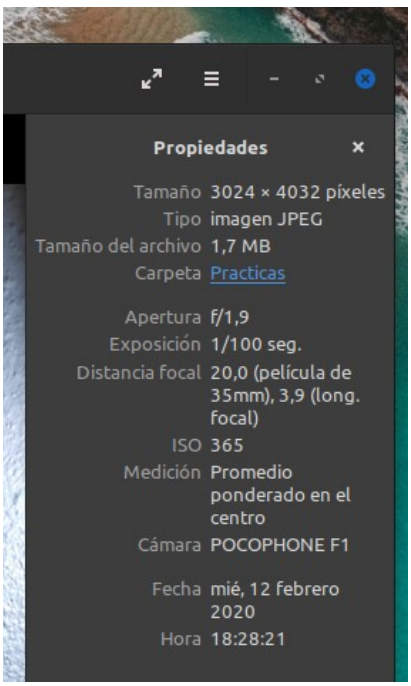


Y se ha ejecutado el script de calibración:

```
(via) j0sem1@j0sem1-Torre-Mint:~/umucv/code/calibrate$ ./calibrate.py "mysmartphone/*.jpg"
processing mysmartphone/IMG_20200327_160607.jpg...
chessboard not found
processing mysmartphone/IMG_20200327_160558.jpg...
ok
processing mysmartphone/IMG_20200327_160612.jpg...
ok
processing mysmartphone/IMG_20200327_160652.jpg...
chessboard not found
processing mysmartphone/IMG_20200327_160552_1.jpg...
ok
processing mysmartphone/IMG_20200327_160600_1.jpg...
ok
processing mysmartphone/IMG_20200327_160623.jpg...
ok
RMS: 1.0179105271991549
camera matrix:
 [[2.89444882e+03 0.00000000e+00 2.02717823e+03]
 [0.00000000e+00 2.89290653e+03 1.51645279e+03]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
distortion coefficients:  [ 1.55795939e-01 -7.51946624e-01 -7.52667371e-04 -1.16049047e-03
  9.49844432e-01]
```

El Focal Length obtenido es **2894’44 píxeles**, un valor muy próximo al obtenido manualmente: 2926 px.

Vamos a comparar el Focal Length obtenido manualmente y con el script de calibración con el que nos proporciona el visor de imágenes Eog. En este pantallazo vemos que obtiene 20 mm. para una película de 35 mm.:



Sabiendo entonces que el ancho del sensor son 36 mm. obtenemos el FOV de la cámara con la misma operación que hemos realizado al principio del ejercicio: FOV = 2 * arc tan ((w/2) / f).

FOV = 2 * arc tan ((36/2) / 20) = **84º**.

Ahora, gracias al FOV y a la resolución de la cámara, obtenemos el Focal Length en píxeles.

FOV = 2 * arc tan ((w/2) / f) ; tan (FOV/2) = (w/2) / f ; f = (w/2) / tan (FOV/2) = (4032/2) / tan (84º/2) = 2016 / 0.9 = 2240 píxeles.

Con este cálculo obtenemos un Focal Length de 2240 píxeles.

Ejercicio CHRO

Se ha utilizado como base el código del Notebook “Chroma” en el que se cambia el fondo de una imagen con un objeto. Comenzaré a relatar los cambios que he realizado sobre dicho código:

- La función que lee una imagen desde un archivo (*readrgb()*) se ha eliminado, se lee la imagen por lo tanto con *cd.imread()*. También se ha eliminado la función *fig()* puesto que no se utilizará.
- Se han creado dos métodos para actualizar los valores del umbral y del tipo de diferencia que se realizará. Estos valores se actualizarán en tiempo real mediante “trackbars”. En un principio se intentó utilizar un botón que actuara de “checkbox” en un panel de control, que cambiara entre RGB y UV, pero al parecer en Python no funciona correctamente, por lo que al final se optó por utilizar un “trackbar” que actuara de interruptor.
- Cuando se pulsa la tecla ‘a’ se captura el fondo y se realiza el chroma. En cualquier momento se puede volver a capturar un fondo diferente pulsando otra vez la tecla.
- Se han eliminado otras muchas cosas del código que no eran útiles o no aportaban nada y hacían el código más complejo y lento. La función fig ralentizaba la imagen y añadía muchísimo retardo, y eliminándola acabamos teniendo el mismo resultado.

He subido a Youtube un vídeo con pruebas realizadas, que se puede ver pulsando aquí.

Iré comentándolo para analizar el comportamiento del chroma. El mismo está dividido en apartados, los cuales pueden verse en la caja de descripción del vídeo.

Durante el primer bloque se utiliza un fondo con un color uniforme y objetos sencillos y con colores uniformes también en su mayoría.

El primer fondo que se utiliza es el verde, utilizado en todos los chromas profesionales. Y **su comportamiento en RGB es bastante bueno**. Separa muy bien los objetos del fondo por los bordes y no confunde colores internos. Con el libro que se utiliza al final al moverlo si se producen interferencias pero nada destacable. **El comportamiento con ultravioleta también es muy bueno**, distinto esto si.

Durante el siguiente trozo de vídeo se testea el fondo de una mesa de escritorio marrón claro. Este color resulta más conflictivo, ya que se confunde más con los objetos y no es completamente uniforme. Además, al no venir la luz de manera uniforme como si ocurría con el color verde (que se utilizaba un monitor para proyectarlo), las sombras no se funden con el fondo y quedan superpuestas. **Con el UV tenemos** ya **un resultado malo**. No distingue bien los objetos y el fondo tampoco es uniforme fuera de ellos. Como

anécdota, cuando se utiliza el Funko de Batman el resultado es mucho peor. **La complejidad no funciona bien**.

El último color uniforme probado es el blanco, y al igual que ocurría con el color verde, **su comportamiento es casi excelente**. En el método RGB cuando se superpone un objeto de color blanco lo confunde con el fondo, algo completamente normal. El UV si que funciona peor que en el caso del color verde. De hecho, como veremos más adelante, el UV solo ha dado un buen comportamiento en ese entorno.

Pasamos ahora al bloque multicolor. Aquí se utilizarán paredes como fondo, y ciertas personas aparecerán actuando de objeto, además moviéndose.

El primer fondo utilizado es una pared rosa lisa, aunque en el vídeo parece tener un color anaranjado. Soy yo mismo el que actúa en esta prueba utilizando ropa oscura. Vemos como **en el caso del RGB la piel se funde con la pared**, ya que el color del fondo es prácticamente igual al de mi piel. Ese es el único problema, ya que la ropa se distingue muy bien incluso cuando hay movimiento. **El UV** aquí **es** sin embargo **desastroso**. No es capaz de dibujar el fondo entero, y me confunde continuamente con la pared.

El segundo fondo es una pared de habitación con una sábana blanca. En este caso hay elementos a los lados, como una mesita, posters o una planta de bambú. Es mi hermana la que aparece esta vez en el vídeo, que utiliza ropa oscura al igual que yo. **Con el RGB funciona decentemente**, incluso no confunde la piel con el fondo, pero al contrario que en el caso anterior la silueta no se distingue claramente y la sombra que produce la persona no se funde con el fondo. **El método UV da otro mal resultado** aquí, aunque es mejor que en el caso anterior, ya que aunque el fondo no es completamente uniforme es muy mejor que en el caso anterior, y la silueta de mi hermana se distingue mejor que la mía.

Como conclusión, el color verde es el que mejor resultado da. Si se tuviera un fondo de color verde uniforme, una luz adecuada que no produzca sombra, y una ropa que no se confunda con el fondo, el resultado sería casi profesional utilizando RGB. Si no se tienen esas condiciones, algo fallará.

Ejercicio HISTCOL

En este ejercicio hay que seleccionar regiones de un vídeo de entrada, guardar dichas regiones seleccionadas, y luego volver a seleccionar la región, para observar si es detectada mediante histogramas.

Se ha utilizado el código para utilizar el ROI (Región de interés) de otro script del repositorio de la asignatura, y cuando se guarda la imagen (pulsando la tecla ‘r’) se guarda dicha imagen con el contorno del roi. Esto es corregible pero no supone problema en la detección y comparación con otras imágenes, ya que todas tendrán dicho contorno.

Hay varias formas de calcular el histograma de una imagen en Python. En este caso se ha utilizado el método de la librería ‘numpy’. Cuando se guarda una imagen, se realiza el histograma de cada canal RGB, y se almacena junto a la imagen. Continuamente se está comprobando la región seleccionada con las imágenes guardadas (si las hubiera), y se muestra en pantalla la diferencia con dichas imágenes. Cuanto mayor sea, menor es la similitud. **Cuando dicha diferencia es menor que las demás y menor que 1.2 se toma como detectada**.

Se muestra además otra ventana que contiene los tres histogramas de la imagen, y se va actualizando conforme se cambia la región seleccionada. Estos histogramas han supuesto mucha dificultad para ser dibujados. En primer lugar hasta no seleccionar un método para calcular el histograma no se sabía como se podrían representar. Al principio se utilizó el método *plt.hist()* de la librería ‘matplotlib.pyplot’. Pero para representar dichos histogramas se abren nuevas ventanas, una para cada histograma, algo muy engorroso. Además, estas se abrían cada frame y no se cerraban, por lo que al cabo de pocos segundos el programa había paralizado el pc.

Finalmente se utilizó la función ***polylines()*** de la librería ‘OpenCV’, que dados una serie de puntos dibuja las líneas que los une. Así se han dibujado tres histogramas a la vez en una ventana a parte, uno para cada canal.

En este vídeo se puede ver una demostración del script, y **funciona casi a la perfección**, detectando todas las regiones que habían sido previamente guardadas.

Ejercicio VROT

En este ejercicio se utilizará la cámara para calcular la velocidad angular a la que rota. Esto se realiza con trayectorias obtenidas por el *tracker* de Lucas-Kanade. Se obtienen puntos son detectados en la imagen, y cuando la cámara se mueve, se dibuja una línea del origen de dichos puntos al punto actual. Esta distancia se mantiene durante 5 frames, después se van borrando los puntos anteriores.

En el centro de la imagen se dibuja otra trayectoria que es la global, representa la media de todas las trayectorias. En la esquina superior izquierda se muestra la cantidad de puntos encontrados, los milisegundos que tarda en tratar la imagen de entrada, la velocidad angular a la que gira la cámara, y la rotación que ha realizado la misma desde que comenzó el script.

La complicación en este ejercicio estaba con la conversión píxeles → grados. **Se utilizó** finalmente **la misma fórmula que en el primer ejercicio para calcular el FOV**: FOV = 2 * arc tan ((w/2) / f). Es la que mejor resultado da, sobre todo cuando se gira 90º. Aunque como se puede apreciar en el vídeo de demostración, cuando gira más no se cuentan correctamente los grados, estos disminuyen. Se de otros compañeros que utilizaron otras fórmulas, como esta de las propiedades de los triángulos: angulo = arc tan (w/f). Al utilizarla yo no conseguí un buen resultado, por lo que la descarté.

Para calcular el arco tangente se utiliza una función de ‘numpy’, pero esta devuelve el ángulo en radianes, que es convertido a grados con la función ‘degrees’ de la librería ‘math’.

Además contamos con **otro problema**, y **es el Focal Lenght utilizado**. En el primer ejercicio se calculó un Focal Lenght dadas unas imágenes de resolución 4032x3024. En este caso se utiliza una resolución de 640x480, por lo que el valor del Focal Lenght cambia. **Para saber qué valor utilizar se volvió a pasar el script ‘calibrate.py’** a la cámara, pero esta vez con unas fotografías realizadas en esta resolución. Estas fotografías se encuentran en la carpeta “chessboard_sift”.

Ejercicio SIFT

En este último ejercicio se leen unas fotografías al principio del script. Son analizadas y se almacenan sus “keypoints”. Después, utilizando la cámara, se ha de reconocer a dichos objetos cuando se enfoque a los mismos, incluso aunque haya reflejos o la imagen no sea clara y perfecta.

Se deben utilizar objetos que sean complejos, tales como carátulas de Cds, carteles, libros, etc. Esto es así porque este programa se basa en el uso de “keypoints”, por lo tanto, cuantos más se encuentren, más identificable será dicha imagen luego.

Se muestran dos ventanas, una con la imagen de entrada que además escribe cuantos puntos se han detectado y el tiempo que se ha tardado en ello, y otra que muestra la imagen con la que se ha encontrado coincidencia si la hay. En esta ventana se muestra el número de matches encontrados y el tiempo que se ha tardado en ello, además del porcentaje de matches. **Se considera que se ha detectado un objeto cuando el porcentaje de matches es mayor que en el resto de casos y este supera el 7%**.

Hablando del vídeo de demostración, **todos los objetos se detectan a la perfección a excepción del cojín con líneas**. Como hemos mencionado antes, este cojín no es complejo y es bastante uniforme, por lo que no se encuentran mactches suficientes y no se detectan. El resto son detectados a la perfección, incluso con malas condiciones.

El funcionamiento del script es sencillo, una vez ejecutado se muestra una ventana en la que se ve lo que escanea la cámara. Además se imprimen el número de puntos (keypoints) que se detectan en la imagen de entrada y el tiempo que han tardado en detectarse en milisegundos.

Este script es el más ineficiente, por lo que cuando **se ejecuta** se hace **en** una **resolución** baja, **320x240**. Además, solo se **comprueba si la entrada “matchea” con algún modelo cuando se pulsa la tecla ‘c’**.