

Prácticos 2022

Redolfi, Javier

7 de marzo de 2022

Evaluación

Ejercicios prácticos para la regularización y aprobación de la materia. Los ejercicios se realizaran en grupos de 2 alumnos. Cada práctico tiene un puntaje. En la tabla se muestran los valores de cada uno de los prácticos.

Número	Tema	Puntaje
Práctico 1	Adivinar	2
Práctico 2	Segmentación	2
Práctico 3	Frame rate	2
Práctico 4	Frame size	2
Práctico 5	Eventos y callbacks	6
Práctico 6	TF. Euclíadiana	6
Práctico 7	TF. Similaridad	6
Práctico 8	TF. Afín	10
Práctico 9	Rectificación	10
Práctico 10	Medición	15
Práctico 11	ARuCo	30
Práctico 12	Alineación con SIFT	10
Práctico 13	Clasificación	20
Práctico 14	Detección	20
Publicación		50
Proyecto		50
Total		241

- Para regularizar hay que sumar 40 puntos o más.
- Para la aprobación hay que sumar 60 puntos o más.

-
- La nota sale del siguiente código python en donde puntos es la cantidad de puntos obtenidos:

```
if(puntos >= 100):  
    nota = 10  
else:  
    nota = int(puntos/10)
```

Práctico 1: Adivinar

- Crear una función **adivinar** que permita adivinar un número generado en forma aleatoria, según las siguientes consignas:
 - El número debe estar entre 0 y 100
 - Este número se genera adentro de la función
 - Además debe recibir un parámetro que sea la cantidad de intentos y en caso de que esta cantidad de intentos sea superada el programa debe terminar con un mensaje
 - Si el usuario adivina antes de superar el número de intentos máximo, se debe imprimir un mensaje con el número de intentos en los que adivinó
- Escribir un programa que pida al usuario que ingrese un número entre 0 y 100 y use la función **adivinar** (todo en el mismo archivo)
- Ejecutar el script desde la consola

Ayuda: código para generar un número aleatorio

```
import random  
numero = random.randint(0, 100)
```

Práctico 2: Segmentación

Segmentando una imagen



- Crear un programa que lea una imagen realice un binarizado de la imagen aplicando un umbral.
- Guarde el resultado en otra imagen.
- No usar ninguna función de las OpenCV, excepto para leer y guardar la imagen.

Ayuda:

- Se puede usar como base el siguiente template:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import cv2

img = cv2.imread('hoja.png', 0)

# Agregar código aquí
# Para resolverlo podemos usar dos for anidados

cv2.imwrite('resultado.png', img)
```

Práctico 3: Frame rate

Considerando el siguiente programa

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import sys
import cv2

if(len(sys.argv) > 1):
    filename = sys.argv[1]
else:
    print('Pass a filename as first argument')
    sys.exit(0)

cap = cv2.VideoCapture(filename)

while(cap.isOpened()):
    ret, frame = cap.read()

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    cv2.imshow('frame', gray)
    if((cv2.waitKey(33) & 0xFF) == ord('q')):
        break

cap.release()
cv2.destroyAllWindows()
```

- ¿Cómo obtener el frame rate o fps usando las OpenCV? Usarlo para no tener que *hard codear* el **delay** del **waitKey**.

Práctico 4: Frame size

Considerando el siguiente programa

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import cv2

cap = cv2.VideoCapture(0)
fourcc = cv2.VideoWriter_fourcc('X', 'V', 'I', 'D')
out = cv2.VideoWriter('output.avi', fourcc, 20.0, (640, 480))

while(cap.isOpened()):
    ret, frame = cap.read()
    if ret is True:
        out.write(frame)
        cv2.imshow('frame', frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    else:
        break

cap.release()
out.release()
cv2.destroyAllWindows()
```

- ¿Cómo obtener el ancho y alto de las imágenes capturadas usando las OpenCV? Usarlo para no tener que *harcodear* el **frameSize** del video generado.

Práctico 5: Eventos y callbacks

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import cv2
import numpy as np

drawing = False # true if mouse is pressed
mode = True # if True, draw rectangle. Press 'm' to toggle to curve
ix, iy = -1, -1

def draw_circle(event, x, y, flags, param):
    global ix, iy, drawing, mode
    if event == cv2.EVENT_LBUTTONDOWN:
        drawing = True
        ix, iy = x, y
    elif event == cv2.EVENT_MOUSEMOVE:
        if drawing is True:
            if mode is True:
                cv2.rectangle(img, (ix, iy), (x, y), (0, 255, 0), -1)
            else:
                cv2.circle(img, (x, y), 5, (0, 0, 255), -1)
    elif event == cv2.EVENT_LBUTTONUP:
        drawing = False
        if mode is True:
            cv2.rectangle(img, (ix, iy), (x, y), (0, 255, 0), -1)
        else:
            cv2.circle(img, (x, y), 5, (0, 0, 255), -1)

img = np.zeros((512, 512, 3), np.uint8)
cv2.namedWindow('image')
cv2.setMouseCallback('image', draw_circle)
while(1):
    cv2.imshow('image', img)
    k = cv2.waitKey(1) & 0xFF
    if k == ord('m'):
        mode = not mode
    elif k == 27:
        break
cv2.destroyAllWindows()
```

Usando como base el programa anterior, crear un programa que permita seleccionar un rectángulo de una imagen, luego

- con la letra “g” lo guardamos a disco en una nueva imagen y salimos
- con la letra “r” restauramos la imagen original y volvemos a realizar la selección
- con la “q” salimos.

Práctico 6: Rotación + Traslación (o Transformación Euclidiana)

Combinar las dos transformaciones anteriores para aplicar una transformación euclidiana (traslación + rotación) a una imagen. Crear un método nuevo que haga esto y que reciba los siguientes parámetros:

- Parámetros

- **angle**: Ángulo
- **tx**: traslación en x
- **ty**: traslación en y

Recordar que la transformación euclídea tiene la siguiente forma:

$$\begin{bmatrix} \cos(\text{angle}) & \sin(\text{angle}) & \text{tx} \\ -\sin(\text{angle}) & \cos(\text{angle}) & \text{ty} \end{bmatrix}$$

Práctico 7: Transformación de similaridad

Agregarle un parámetro al método anterior para permitir también el escalado de la imagen.

- Parámetros

- **angle**: Ángulo
- **tx**: traslación en x
- **ty**: traslación en y
- **s**: escala

Recordar que la transformación de similaridad tiene la siguiente forma:

$$\begin{bmatrix} s \cdot \cos(\text{angle}) & s \cdot \sin(\text{angle}) & tx \\ -s \cdot \sin(\text{angle}) & s \cdot \cos(\text{angle}) & ty \end{bmatrix}$$

Práctico 8: Transformación afín - Incrustando imágenes

Teniendo en cuenta que:

- Una transformación afín se representa con una matriz de 2×3
- Tiene **6** grados de libertad y puede ser recuperada con **3** puntos

Usando como base el programa escrito en la práctica 5, se pide:

- Crear un programa que permita seleccionar con el mouse 3 puntos de una primera imagen.
- Luego crear un método que compute una transformación afín entre las esquinas de una segunda imagen y los 3 puntos seleccionados.
- Por último aplicar esta transformación sobre la segunda imagen, e incrustarla en la primera imagen.

Ayuda

- **cv2.getAffineTransform**
- **cv2.warpAffine**
- Generar una máscara para insertar una imagen en otra

Práctico 9: Rectificando imágenes

Teniendo en cuenta que:

- Una homografía se representa con una matriz de 3×3
- Tiene **8** grados de libertad y puede ser recuperada con **4** puntos

Usando como base el programa anterior, se pide:

- Crear un programa que permita seleccionar 4 puntos de una primera imagen.
- Luego crear un método que compute una homografía entre dichos 4 puntos y una segunda imagen estándar de $m \times n$ pixeles.
- Por último aplicar esta transformación para llevar (rectificar) la primera imagen a la segunda de $m \times n$.

Ayuda

- `cv2.getPerspectiveTransform`
- `cv2.warpPerspective`

Práctico 10: Medición de objetos

La siguiente imagen es una foto de la fachada de una casa.



Figura 1: Foto de ejemplo. Sacar una foto propia.

Como dato de entrada, sabemos que el tamaño del marco de la puerta es de 2.10m de alto por 0.73m de ancho. En base a estas medidas hacer un programa que permita medir el largo de cosas en el plano de la fachada.

Capturar una imagen de una fachada con perspectiva y hacer un programa que permita:

- encontrar la transformación perspectiva entre los 4 vértices del marco y un rectángulo con la misma relación de aspecto que la puerta real;
- aplicar dicha transformación a la imagen y mostrarla en una ventana;
- sobre esta ventana permitir que el usuario haga dos clicks y mostrar la distancia en metros entre dichos puntos;
- permitir que cuando se presione la tecla “r” se reinicie la medición;
- por último medir dos objetos en la imagen (ventana, casilla del gas, etc.) y comparar los resultados con la medida real.

Práctico 11: Práctico Libre ArUCo

En base al teórico de ArUCo y a los videos presentados, se propone realizar una aplicación simple en base a ArUCo. Puede ser por ejemplo realidad virtual en 2D o en 3D, estimación de posición y orientación de objetos en movimiento, seguimiento de objetos, o simplemente hacer instalar algún repositorio interesante y aplicarlo a algo.

- <https://www.youtube.com/watch?v=nsu9tNIJ6F0>
- https://youtu.be/VsIMl8O_F1w?t=55
- <https://www.youtube.com/watch?v=qqDXwKDr0vE>
- <https://www.youtube.com/watch?v=RNwTGPvuhPw&t=8>
- <https://www.youtube.com/watch?v=xzG2jQfxLlY>
- <https://www.youtube.com/watch?v=jwu9SX3YPSk>
- <https://www.youtube.com/watch?v=FB14Y55V2Z4>

El puntaje del práctico dependerá de la complejidad, utilidad o nivel de innovación de la aplicación (como máximo 30 puntos).

Práctico 12 - Alineación de imágenes usando SIFT

1. Capturar dos imágenes con diferentes vistas del mismo objeto
2. Computar puntos de interés y descriptores en ambas imágenes
3. Establecer matches entre ambos conjuntos de descriptores
4. Eliminar matches usando criterio de Lowe
5. Computar una homografía entre un conjunto de puntos y el otro
6. Aplicar la homografía sobre una de las imágenes y guardarla en otra (mezclarla con un alpha de 50 %)

Usar el siguiente código como ayuda:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import numpy as np
import cv2

MIN_MATCH_COUNT = 10

img1 = # Leemos la imagen 1
img2 = # Leemos la imagen 2

dscr = # Inicializamos el detector y el descriptor

kp1, des1 = # Encontramos los puntos clave y los descriptores con SIFT en la imagen 1
kp2, des2 = # Encontramos los puntos clave y los descriptores con SIFT en la imagen 2

matcher = cv2.BFM Matcher(cv2.NORM_L2)

matches = matcher.knnMatch(des1, des2, k=2)

# Guardamos los buenos matches usando el test de razón de Lowe
good = []
for m, n in matches:
    if m.distance < 0.7*n.distance:
        good.append(m)

if len(good) > MIN_MATCH_COUNT:
    src_pts = np.float32([kp1[m.queryIdx].pt for m in good]).reshape(-1, 1, 2)
    dst_pts = np.float32([kp2[m.trainIdx].pt for m in good]).reshape(-1, 1, 2)

    H, mask = cv2.findHomography(dst_pts, src_pts, cv2.RANSAC, 5.0) # Computamos la homografía con RANSAC

wimg2 = # Aplicamos la transformación perspectiva H a la imagen 2

# Mezclamos ambas imágenes
alpha = 0.5
blend = np.array(wimg2 * alpha + img1 * (1 - alpha), dtype=np.uint8)
```

Ejemplo de alineación:



Figura 2: Puntos claves en la imagen 1



Figura 3: Puntos claves en la imagen 2



Figura 4: Matches entre ambas imágenes



Figura 5: Mezcla de ambas imágenes

Práctico 13 - Clasificación de imágenes usando CNN

En este práctico vamos a implementar un clasificador de imágenes usando CNN. El objetivo sería diseñar y entrenar un algoritmo que en forma automática clasifique una imagen como correspondiente a una clase entre varias. Las clases las vamos a elegir nosotros (perros y gatos, autos y motos, o algo más interesante o útil).

Para probar el modelo, se deben buscar 2 imágenes nuevas, mostrarlas en pantalla (matplotlib), clasificarlas con el modelo y mostrar el resultado de la clasificación (score para cada clase).

Usar template disponible en colab.

Práctico 14 - Detección de objetos usando CNN

El objetivo de este práctico es entrenar un detector de objetos en imágenes usando la red Yolo.

Para probar el modelo buscar 2 imágenes nuevas, mostrarlas en pantalla (matplotlib), detectar objetos con el modelo y mostrar el resultado de la detección (recuadro que indique la ubicación y clase del objeto).

Usar template disponible en colab.

Publicación

Realizar una publicación en un congreso como por ejemplo las jornadas de ciencia y tecnología que se realizarán en nuestra facultad o cualquier otro congreso.

Proyecto

Realizar un proyecto que use visión por computadora: proyecto final, tesis o aplicación.