Prácticos 2025

16 de abril de 2025

Evaluación

Ejercicios prácticos para la regularización y aprobación directa de la materia. Cada práctico tiene un puntaje máximo asignado que se muestran en la tabla a continuación.

Número	Tema	Puntaje
Práctico 1	Adivinar	5
Práctico 2	Segmentación	5
Práctico 3	Videos	5
Práctico 4	Eventos y callbacks	5
Práctico 5	Rotación y traslac.	5
Práctico 6	Similaridad	5
Práctico 7	Transformación Afín	10
Práctico 8	Rectificación	10
Práctico 9	Medición	20
Práctico 10	ARuCo	30
Práctico 11	Clasificación	15
Práctico 12	Detección	15
Práctico 13	Caso de Estudio	20
Publicación	Publicación	50
Proyecto	Proyecto final	50

Para la aprobación hay que sumar 60 puntos o más y completar las presentaciones individuales de los avances de trabajo en el laboratorio.

Práctico 1 - Función en python

1. Crear una función **`adivina`** que permita adivinar un número secreto generado en forma aleatoria, según las siguientes consignas:
   - El número secreto debe estar entre 0 y 100, y debe ser generado dentro de la función.
   - La función **`adivina`** debe recibir un parámetro que indique la cantidad de intentos permitidos.
2. Luego escribir un programa **`adivinador.py`** que:
   - pida al usuario que adivine el número secreto, es decir que ingrese un número entre 0 y 100,
   - use la función **`adivina`** anterior para verificar si adivinó (todo en el mismo archivo),
   - si el usuario adivinó el número secreto antes de superar la cantidad permitida de intentos, imprima un mensaje con el número de intentos en los que adivinó,
   - en caso de que esta cantidad de intentos sea superada el programa debe terminar con un mensaje.
3. Finalmente ejecutar el programa **`adivinador.py`** desde la consola.

Ayuda: código para generar un número aleatorio

```
import random

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

Práctico 2 - Segmentando una imagen

1. Escribir un programa en python que lea una imagen y realice un umbralizado binario, guardando el resultado en otra imagen.
   - NOTA: No usar ninguna función de las OpenCV, excepto para leer y guardar la imagen.

Ayuda:

Figura 1: Binarizado de imagen.

- 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 - Propiedades de video

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)

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

delay = 33
while(cap.isOpened()):
    ret, frame = cap.read()
    if ret is True:
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        out.write(gray)
        cv2.imshow('Image gray', gray)
        if cv2.waitKey(delay) & 0xFF == ord('q'):
            break
    else:
        break

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

1. Cómo obtener el número de cuadros por segundo o fps de un video guardado usando las OpenCV? Usar luego para no tener que *harcodear* el **`delay`** del **`waitKey`**, es decir que el programa muestre el siguiente cuadro con el retardo de tiempo que corresponde a como fue grabado.

2. Cómo obtener el ancho y alto de las imágenes capturadas usando las OpenCV? Usar para dar dimensiones al video generado, es decir para no tener que *harcodear* el **`frameSize`**.

Práctico 4 - Eventos y callbacks

Considere el siguiente código.

```
#! /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()
```

1. Usando como base el código anterior, escribir un programa que permita seleccionar una porción rectangular de una imagen, luego
   - con la letra "g" guardar la porción de la imagen seleccionada como una nueva imagen,
   - con la letra "r" restaurar la imagen original y permitir realizar una nueva selección,
   - con la "q" finalizar.

Práctico 5 - Rotación + Traslación (o Transformación Euclidiana)

1. Crear una función que aplique una transformación euclidiana, recibiendo los siguientes parámetros:
   - Parámetros
     - **`angle`**: Ángulo
     - **`tx`**: traslación en x
     - **`ty`**: traslación en y

   Recordar que la transformación euclidiana tiene la siguiente forma:

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

2. Usando como base el programa anterior, escribir un programa que permita seleccionar una porción rectangular de una imagen y

   - con la letra "e" aplique una transformación euclidiana a la porción de imagen seleccionada (solicitando al usuario los tres parámetros, ángulo, traslación en x y traslación en y) y la guarde como una nueva imagen.

Práctico 6 - Transformación de similaridad

1. Agregar a la función anterior un parámetro que permita aplicar un escalado a la porción rectangular de imagen.

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

   Recordar que la transformación de similaridad tiene la siguiente forma:

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

2. Luego, usando como base el programa anterior, escribir un programa que permita seleccionar una porción rectangular de una imagen y

   - con la letra "s" aplique una transformación de similaridad a la porción de imagen seleccionada (solicitando al usuario los cuatro parámetros, escala, ángulo, traslación en x y traslación en y) y la guarde como una nueva imagen.

Práctico 7 - 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 no colineales.

1. Crear una función que compute la transformación afín entre 3 pares de puntos correspondientes.

2. Usando como base el programa anterior, escriba un programa que

    - con la letra "a" permita seleccionar con el mouse 3 puntos no colineales en una imagen e incruste entre estos puntos seleccionados una segunda imagen.

Ayuda

- **`cv2.getAffineTransform`**
- **`cv2.warpAffine`**
- Generar una máscara para insertar una imagen en otra
- Un ejemplo para usar esta transformación es para aplicar el estilo *Italico* a un texto (ver fig. 2.

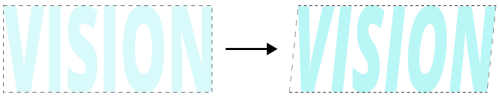

Figura 2: Ejemplo de transformación afín.

Práctico 8 - Rectificando imágenes

Teniendo en cuenta que:

- una homografía se representa con una matriz de 3×3 (pero tiene sólo **`8`** grados de libertad) y
- puede ser recuperada con **`4`** puntos no colineales.

1. Crear una función que compute la homografía entre los 4 pares de puntos correspondientes.
2. Usando como base el programa anterior, escriba un programa que

    - con la letra "h" permita seleccionar con el mouse 4 puntos no colineales en una imagen y transforme (rectifique) la selección en una nueva imagen rectangular.

En detalle se deberá hacer:

- 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 porción de la primera imagen definida por los 4 puntos elegidos a la segunda de $m \times n$.

Ayuda

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

Práctico 9 - Medición de objetos sobre plano calibrado

La rectificación de la distorsión perspectiva en de una imagen de un objeto plano permite relacionar en forma lineal (mediante una escala adecuada) el tamaño final de la imagen rectificada con las dimensiones reales del objeto original, y de todos los objetos que comparten el mismo plano. Si además se conocen las dimensiones de uno de los objetos en el plano entonces se puede determniar la escala real entre las dimensiones en píxeles de la imagen y las dimensiones reales de los objetos en el mismo plano.

Teniendo en cuenta lo anterior y como práctica del tema se pide:

- capturar una imagen propia con perspectiva, que puede ser de una fachada o cualquier otra escena plana, que tenga un objeto rectangular de dimensiones conocidas,
- escribir un programa que permita encontrar la transformación perspectiva entre los 4 vértices del objeto en la imagen y un rectángulo con la misma relación de aspecto que el objeto real,
- que luego de elegir los 4 vértices aplique dicha transformación a la imagen y la muestre en una nueva ventana "calibrada para medición",
- sobre esta ventana de medición el programa debe permitir que el usuario elija con el ratón dos puntos cualquiera y debe mostrar la distancia en metros entre dichos puntos;
- la ventana de medición debe poder ser restaurada cuando se presione la tecla "r" para realizar una nueva medición;
- por último medir dos objetos en la imagen (ventana, casilla del gas, etc.) y verificar los resultados con la medida real.

VARIANTE: Teniendo en cuenta la preservación de la relación cruzada entre segmentos de una transformación perspectiva (ver teórico), escribir un programa que permita determinar distancias reales sobre el plano de un escena, utilizando como dato una o dos distancias conocidas de sobre una línea del mismo plano.

Práctico 10 - 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=FBl4Y55V2Z4

El puntaje del práctico dependerá de la complejidad, utilidad o nivel de innovación de la aplicación.

Práctico 11 - Clasificación de imágenes usando CNN

En este práctico vamos a implementar un clasificador de imágenes usando CNN. El objetivo será 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 (puntaje para cada clase).

Usar plantilla disponible en colab.

Práctico 12 - 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.

Práctico 13 - Caso de estudio en una empresa

En este práctico se deberá seleccionar una empresa a la cual se tenga acceso y en dicha empresa se deberán investigar que problemas se pueden resolver o que mejoras se pueden realizar usando técnicas de visión por computadora, presentando un informe detallado de lo investigado y una o más propuestas de solución.

El informe deberá contemplar lo siguiente:

- indicar que algoritmos se podrían utilizar para resolver el problema justificando la propuesta (con aplicaciones que ya lo realizan o con el sustento teórico correspondiente),
- armar un diagrama en bloques en donde se distingan el orden de las diferentes etapas del algoritmo y
- seleccionar el hardware necesario para correr el algoritmo en planta junto con un estimativo de costos del mismo.

Práctico 14 - Publicación/presentación

Realizar una publicacación/presentación en un congreso nacional.

Práctico 15 - Proyecto

Realizar un proyecto que use visión por computadora: proyecto final, tesis o aplicación propuesta en el Caso de estudio.