Herramientas Software para Tratamiento de Imágenes
Máster Oficial en Visión Artificial
Hoja de Problemas Evaluable

Instrucciones para la entrega

Una vez resueltos los problemas enunciados a continuación, el alumno utilizará la tarea habilitada en aula virtual para subir los ficheros fuente en un fichero zip. El nombre del fichero comprimido deberá formarse siguiendo el siguiente esquema: "Apellido1_Apellido2_Nombre.zip". Cada solución debe almacenarse en una carpeta cuyo nombre indique el número de ejercicio resuelto: "ejercicio9", "ejercicio10", …

Ejercicio 9

Desarrollar un método para Open3D que permita filtrar outliers en una nube de puntos 3D. Existen diferentes tipos de filtrado de outliers en la literatura, en este ejercicio centraremos nuestra atención en el filtrado de outliers por radio (explicado en clase, módulo Python++).

Para desarrollar una versión eficiente del método se deberá hacer uso de estructuras KDtrees. El método de filtrado de outliers desarrollado debe responder a la siguiente interfaz:

```
filtered_point_cloud = radius_outlier_removal
            (point_coud, min_number_points, radius)
```

Así mismo, y de cara a comprobar el correcto funcionamiento del método desarrollado, se debe presentar un programa en python que pinte el resultado y responda a la siguiente interfaz:

```
python outlier_removal.py --ipc=PLYPointCloud --points=16
              --radius=0.05 --opc=resulting_cloud.pcd
```

Los posibles nombres de nubes de puntos que se desean procesar son `PLYPointCloud` y `EaglePointCloud`. Cada una de ellas corresponde a uno de los datasets de Open3d (link).

Por ejemplo, si el valor para la nube de puntos es `--ipc=PLYPointCloud`, el programa debe cargar la siguiente nube de puntos `open3d.data.PLYPointCloud()`.

Ejercicio 10

Se desea mejorar el rendimiento del ejercicio 7 haciendo uso de alguna de las estrategias de paralelización explicadas en clase. El enfoque propuesto en dicho ejercicio exige comparar la imagen de entrada con todas las imágenes del dataset. Este proceso puede paralelizarse, reduciendo así el tiempo de ejecución.
Se deja a elección del alumno la estrategia y los módulos a usar: `numba` y/o, `multiprocessing`. Se debe reportar un pequeño análisis de tiempos y de factor de aceleración de ambos enfoques (con y sin paralelización). Para ello, se pide montar un `jupyter notebook` con texto narrativo, código y gráficos de rendimiento. Utilizar las celdas de texto narrativo para explicar el tipo de tecnologías y estrategias empleadas.

Ejercicio 11

Portar a Matlab el Ejercicio 3 de la Hoja de Problemas I: análisis del rendimiento de un algoritmo de reconstrucción 3D.

Ejercicio 12

Javier se ha comprado un disco NAS para que su familia pueda subir todas las fotos que van haciendo de su hija Ana con sus móviles. Javier quiere tener todas las fotos posibles de su hija, pero no con cualquier calidad. Le gustaría poder eliminar todas aquellas fotos que estén desenfocadas de su biblioteca. Nuestro objetivo es ayudar a Javier a automatizar esta tarea.

Lo primero que haremos será generar una colección de fotos enfocadas y desenfocadas que nos permitan probar el método que vamos a desarrollar. Para ello, se pide primero crear una función (`blurrer`) en Matlab que reciba como argumentos un directorio de entrada y un directorio de salida. El directorio de entrada deberá contener una colección de imágenes inicial. Tras llamar a nuestra función, el directorio de salida contendrá esas mismas imágenes, pero desenfocadas. La función suavizará las imágenes de entrada haciendo uso de la función `imsmooth` y utilizando diferentes tamaños de kernel 5x5, 15x15, 21x21.

Una vez generada la colección de imágenes de prueba podemos desarrollar nuestro detector de imágenes desenfocadas. Para ello crearemos una segunda función en Matlab que llamaremos `blurdetection`, y recibirá como argumento de entrada un directorio. Esta función irá presentando en una ventana las imágenes que contiene el directorio (una por una). Sobre la imagen deberá aparecer un texto, indicando si la imagen está o no desenfocada y que “cantidad” de desenfoque presenta (`figure`, `text`, `waitforbuttonpress`). Pero... ¿cómo podemos cuantificar el desenfoque de una foto?

Una imagen desenfocada presenta un bajo porcentaje de altas frecuencias. Existen diferentes métodos en la literatura que permiten medir el nivel de desenfoque de una imagen. Nosotros utilizaremos un método basado en la varianza de la Laplaciana. Para ello seguiremos los siguientes pasos:

1. Convolucionar la imagen con un kernel laplaciano (`fspecial`)
2. Calcular la varianza de la imagen resultante
    - La varianza cuantifica el nivel de desenfoque de la imagen (menor varianza implica mayor desenfoque)
    - En principio será el usuario el encargado de determinar utilizando este valor qué es para él una imagen enfocada y qué es para él una imagen desenfocada. Por defecto fijaremos el umbral en 90.

Ejercicio 13

Desarrollar un sistema de visión artificial que permita hacer seguimiento visual por color de un objeto. El sistema debe ser capaz de detectar el objeto (en el caso de que se encuentre presente) en cada fotograma. El objeto puede moverse libremente, pudiendo cambiar su escala y la perspectiva con la que se ve el mismo. Para ello utilizaremos algunas de las rutinas del paquete de procesamiento de imagen de Matlab.

Lo primero que necesitamos es ser capaces de segmentar el objeto a seguir en cada fotograma. Para ello convertiremos cada fotograma al espacio de color HSV (`rgb2hsv`). Dado el rango de color en el que se mueve nuestro objeto, segmentaremos el fotograma generando una imagen binaria.

Una vez determinado el rango HSV de nuestro objeto, definiremos una función llamada color_tracking que se llevará el peso del seguimiento. Por cada fotograma del vídeo se llevarán a cabo las siguientes operaciones:

1. Aplicar un filtro gausiano al fotograma para eliminar ruido (`imsmooth`)
2. Convertir el fotograma al espacio de color HSV (`rgb2hsv`)
3. Segmentar el fotograma utilizando el rango de color seleccionado
4. Eliminar pequeños focos de ruido en la imagen resultante:
    - Aplicar dos operaciones de erosión consecutivas (`strel`, `imerode`), utilizando un elemento estructurante cuadrado de tamaño 3x3.
    - Aplicar dos operaciones de dilatación consecutivas, (`strel`, `imdilate`), utilizando un elemento estructurante cuadrado de tamaño 3x3.
5. Detectar los distintos contornos que aparecen en la imagen y quedarse con aquel que presente mayor área (`regionprops`).
6. Presentar en pantalla la imagen con el contorno sobreimpreso en color verde. Pintar también el centroide del contorno en color rojo.
7. Esperar 33ms (`pause`) y continuar con el siguiente fotograma.

Ejercicio 14

Desarrollar un programa en Matlab que permita detectar códigos de barras utilizando rutinas básicas de procesamiento de imagen. Para probar el algoritmo se proporciona un pequeño conjunto de imágenes junto con el enunciado. Igualmente, el alumno podrá capturar sus propias imágenes y comprobar la efectividad del método desarrollado. Los pasos se detallan a continuación:

1. Reducir la resolución de la imagen a una cuarta parte. No necesitamos imágenes de 8 Mb (como las del dataset) para detectar códigos de barras.
2. Convertir la imagen a escala de grises
3. Calcular la magnitud del gradiente para la imagen en escala de grises.
    - Usar el operador Scharr para calcular el gradiente en la dirección x e y
    - El resultado serán dos imágenes (`gradientX`, `gradientY`). Una con el gradiente para la dirección x y otra con el gradiente en la dirección y.
    - Calcular la diferencia en valor absoluto de ambas imágenes `|gradientX - gradientY|`
4. Para eliminar ruido en la imagen de gradiente resultante, aplicar un filtro media con tamaño de kernel 9x9 por ejemplo
5. A continuación, umbralizar el fotograma resultante. Buscamos mantener la atención únicamente en aquellas zonas con un gradiente elevado. Todo valor de gradiente por debajo de 200 tomará valor cero. Todo valor por encima de 200 tomará valor 1.
6. Aplicar una operación de cierre morfológico (`imclose`) utilizando un elemento estructurante de 21x7. El objetivo es eliminar las separaciones existentes entre las barras y simplificar así la tarea de extraer el contorno del código de barras.
7. Aplicar cuatro operaciones de erosión y cuatro de dilatación empleando un elemento estructurante cuadrado de 3x3. El objetivo es eliminar pequeños blobs en la imagen.
8. Calcular los contornos en la imagen resultante tras aplicar las operaciones morfológicas. Seleccionar aquel de mayor tamaño y dibujar el bounding box que lo contiene en la imagen original.

El programa desarrollado deberá presentar al menos una función llamada barCodeDetector que reciba como argumento una imagen de entrada y presente en pantalla el resultado de la detección.