Exercise 3: Pyramids and Optic Flow

Due date: 16/05/2021

In the lectures we discussed optic flow, in this exercise you will implement the Lucas Kanade algorithm. All functions should be able to accept both gray-scale and color images. In the optical-flow methods, you should accept a color image, but work on the gray-scale copy of the image.

1 Lucas Kanade optical flow

Write a function which takes an image and returns the optical flow by using the LK algorithm.

```
def opticalFlow(im1: np.ndarray, im2: np.ndarray, step_size=10, win_size=5)
                                                    -> (np.ndarray, np.ndarray):
    """
    Given two images, returns the Translation from im1 to im2
    :param im1: Image 1
    :param im2: Image 2
    :param step_size: The image sample size:
    :param win_size: The optical flow window size (odd number)
    :return: Original points [[x,y]...], [[dU,dV]...] for each points
    """
```

In order to compute the optical flow, you will first need to compute the gradients I_x and I_y and then over a window centered around each pixel we calculate

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -\sum I_x I_t \\ -\sum I_y I_y \end{bmatrix}$$

Remember(1) the checks for the $\lambda_{1/2}$:

- $\lambda_1 \geq \lambda_2 > 1$

- $\frac{\lambda_1}{\lambda_2} < 100$

For some points you will not be able to get a good optical flow, meaning they won't pass the λ constraints. That is why you need to return only the points that are good, since only they will have a matching $\Delta X, \Delta y$. Remember(2) **LK algorithm is used for sub-pixel movements!**

The best way to check your self is to take an image, apply a transformation (i.e. translation) and test. Use the method *displayOpticalFlow* to look at the results, you can also check the mean of the $\Delta X, \Delta Y$ to see that it matches the initial transformation. Even if you had a simple translation, not all points will give the same optical flow, there will be some outliers (*"I accept chaos, I'm not sure whether it accepts me." — Bob Dylan*)

2 Gaussian and Laplacian Pyramids

Remarks:

1. For all pyramids, use Gauusian kernel with a size of 5×5 and $\sigma = 0.3 * ((k_{size} - 1) * 0.5 - 1) + 0.8$ (you can use cv2.getGaussianKernel but make sure to read the docs!!).
2. Remember, the sum of the kernel when down-sampling should be 1, and when up-sampling the sum should be 4.
3. Each level in the pyramids, the image shape is cut in half, so for x levels, crop the initial image to $2^x \cdot \lfloor img_{size}/2^x \rfloor$

2.1 Gaussian Pyramids

Write a function that returns a Gaussian pyramid for a given image.

```
def gaussianPyr(img: np.ndarray, levels: int = 4) -> List[np.ndarray]:
    """
    Creates a Gaussian Pyramid
    :param img: Original image
    :param levels: Pyramid depth
    :return: Gaussian pyramid (list of images)
    """
```

The following function should receive a layer from a Gaussian pyramid and expand it using the provided kernel, one level up.

```
def gaussExpand(img: np.ndarray, gs_k: np.ndarray) -> np.ndarray:
    """
    Expands a Gaussian pyramid level one step up
    :param img: Pyramid image at a certain level
    :param gs_k: The kernel to use in expanding
    :return: The expanded level
    """
    pass
```

2.2 Laplacian Pyramids

Write a function that returns a Laplacian pyramid for a given image

```
def laplaceianReduce(img: np.ndarray, levels: int = 4) -> List[np.ndarray]:
    """
    Creates a Laplacian pyramid
    :param img: Original image
    :param levels: Pyramid depth
    :return: Laplacian Pyramid (list of images)
    """
def laplaceianExpand(lap_pyr: List[np.ndarray]) -> np.ndarray:
    """
    Resotrs the original image from a laplacian pyramid
    :param lap_pyr: Laplacian Pyramid
    :return: Original image
    """
```

2.3 Pyramid Blending

```
def pyrBlend(img_1: np.ndarray, img_2: np.ndarray, mask: np.ndarray, levels: int) ->
                                                                    (np.ndarray, np.ndarray):
    """
    Blends two images using PyramidBlend method
    :param img_1: Image 1
    :param img_2: Image 2
```

```
:param mask: Blend mask
:param levels: Pyramid depth
:return: (Naive blend, Blended Image)
"""
```

The Naive blend, is blending without using the pyramid.

3 Important Comments

- The input of all the above functions can be either color or gray-scale images.
- Submit the two python files *ex3_main.py,ex3_utils.py* and any other python files you feel are necessary to use, along with all the images you use in your code.
- Your code should work 'out of the box', make sure your paths are correct and no other libraries are added.
- Do not change the file *ex3_main.py* except change the images path if you feel creative, and please do!!
- NO RAR or ZIP files!