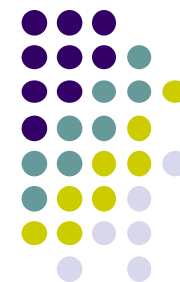# Image Processing and Computer Vision - Lab 4

Roberta Macaluso

Politecnico di Torino

Dipartimento di Automatica e Informatica (DAUIN)

Torino - Italy
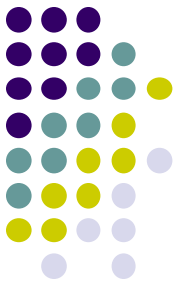
roberta.macaluso@polito.it

# The Plan

1. Intro to Image Processing
2. Intro to OpenCV
3. Fourier Transform (and Friends)
   - today (23/04) and next week (30/04)
4. Image Segmentation
5. CCD, CMOS, and Optical Systems
6. Car Lane Detection
7. Face Detection and Tracking
8. Neural Network Introduction
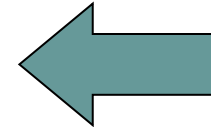
# Fourier Transform

- Today and next week
  - 3 hours
- Text of the exercises/tasks
  - on the Teaching Portal
- Today: you need two still images
  - get them from the Teaching Portal
- Next week: another image (choose freely)
- Goal
  - Check the theory behind the Fourier transform (and its inverse)
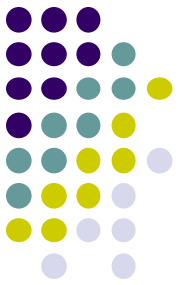
# **Fourier Transform**

- Four exercises
  1. apply the discrete Fourier transform (DFT) to a sin function image and a circle image; then anti-transform both
  2. apply the DFT to an image of your choice, and add a high-pass filter as a mask
  3. apply the DFT to an image of your choice, and add a low-pass filter as a mask
  4. apply the DFT to an image of your choice, and add a band-pass filter as a mask
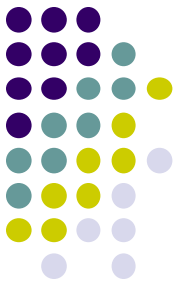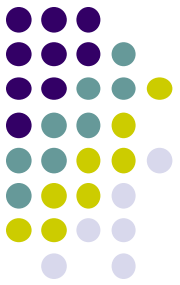
# DFT

```
complex_image_result = cv2.dft(complex_image_src,
                flags=cv2.DFT_COMPLEX_OUTPUT)
```

where both the source and destination images are <u>complex</u> images

To have a "normal" (source) image as a complex one you should:

- `np.float32(img)`
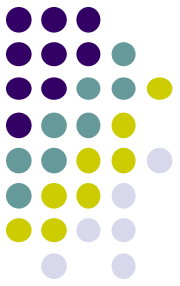
# **Show the DFT result**

- DFT output image is complex
- We can print/visualize its magnitude, only!

```
magnitude = cv2.magnitude(real_part,
                imaginary_part)
```

where:

- the real and imaginary part are the first and second "channel" of the complex image (DFT output)
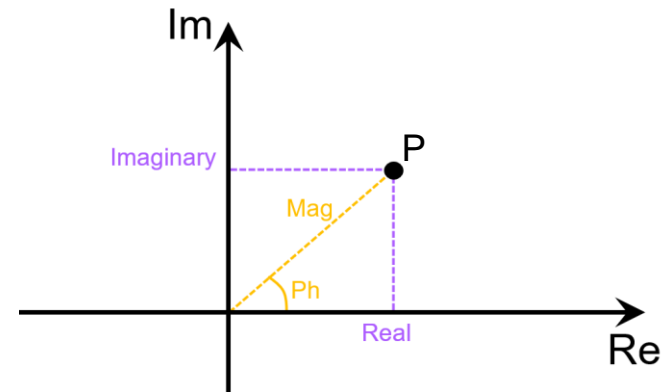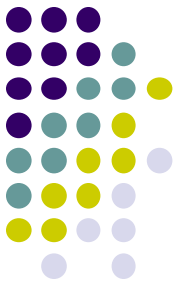
# Cartesian and Polar Coordinates

- Given the real and imaginary parts, we can pass from cartesian coordinates to polar ones in order to get magnitude and phase.

```
magnitude, phase =
cv2.cartToPolar(real, imaginary)
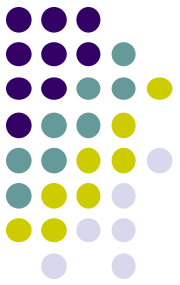```



- Or get the inverse function:

```
real, imaginary =
cv2.polarToCart(magnitude, phase)
```
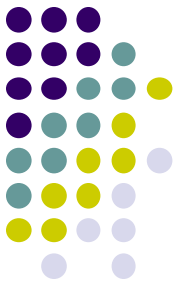
# **Visualize the magnitude**

- Values in the magnitude matrix can be in different orders of magnitude

  - … we need to move them in a log scale

  - `np.log()`

- Also, the image must be shifted to the center, to analyze its spectrum

  - its four quadrants are inverted, by default!
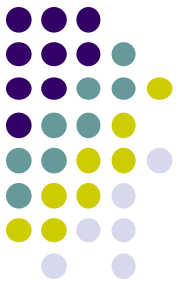
# Inverse DFT

```
complex_image_dst =
cv2.idft(complex_image_src)
```

where both the source and destination images are <u>complex</u> images, as before

# *-band filters

- First, compute the DFT, up to the magnitude (as before)
- Then, create the mask/filter:
    - `np.zeros(), np.ones()` can be useful to get started
    - compute the circular area(s) in the middle of the mask (this could be a bit tricky; it involves numpy quite heavily)
        - the last two links in the text of the lab can really help on this
    - apply the mask by multiplying (`*`) the DFT output and the mask
    - finally, perform the inverse DFT (as before)

# **Fourier Transform**

- In the first session: you should be able to complete the first exercise

- Hints, insights, links, etc. are in the text of the exercises

  - I am here for you…

  - … please ask if you need any help or clarification

# License
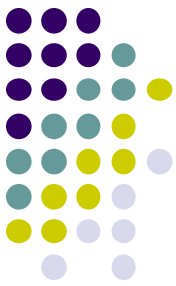
This work is licensed under the Creative Commons "Attribution-NonCommercial-ShareAlike International (CC BY-NC-SA 4.0)" License.

You are free to:

- **Share** - copy and redistribute the material in any medium or format
- **Adapt** - remix, transform, and build upon the material

for any purpose, even commercially.

Under the following terms:

- **Attribution** - You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
- **Noncommercial** - You may not use the material for commercial purposes.
- **Share Alike** - If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.