



UNIVERSITÀ DI PARMA

# Laboratory #05

## Edges

- Per l'esercizio proposto è possibile utilizzare il template `at<>()` di `cv::Mat`
- Esempi:
  - `image.at<float>(r,c)` restituisce direttamente il float in posizione `r,c`
  - `image.at<uint8_t>(r,c)` restituisce direttamente un unsigned char in posizione `r,c`
  - `image.at<cv::Vec3b>(r,c)[k]` restituisce direttamente il canale `k` di un'immagine a 3 canali unsigned char
  - ...

- Realizzare una funzione che genera un kernel Gaussiano per smoothing verticale
  - `void gaussianKrnl(float sigma, int r, cv::Mat& krnl)`
    - Krnl: singolo canale float32
    - Sigma: deviazione standard gaussiana
    - R: raggio del kernel, quindi il kernel avrà dimensione  $(2 \cdot r + 1) \times 1$
    - Krnl: risultato normalizzato (ogni elemento va diviso per la somma dei pesi)

- Applicare quanto visto precedentemente e le regole di separabilità dei filtri gaussiani per realizzare una funzione di smoothing gaussiano bidimensionale
  - Sfruttare la `myfilter2D()` sviluppata precedentemente o la `filter2D()` di OpenCV
  - `void GaussianBlur(const cv::Mat& src, float sigma, int r, cv::Mat& out, int stride=1)`
  - Visualizzare il risultato
    - Per Debug visualizzare anche i risultati intermedi

- Basandosi su quanto già sviluppato, scrivere una funzione che calcoli magnitudo e orientazione di un Sobel 3x3
  - `void sobel3x3(const cv::Mat& src, cv::Mat& magn, cv::Mat& or)`
    - Src: immagine in ingresso uint8
    - Magn: magnitudo risultato float32
    - Or: orientazione float32

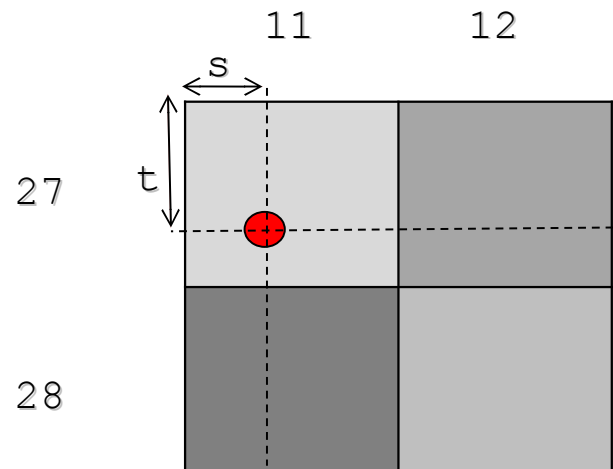
- Visualizzare magnitudo e orientazione calcolate nei passi precedenti applicati al risultato del filtro gaussiano
- Per l'orientazione potete basarvi sul codice seguente:
  - `cv::Mat adjMap;`
  - `cv::convertScaleAbs(orientation, adjMap, 255 / (2*M_PI));`
  - `cv::Mat falseColorsMap;`
  - `cv::applyColorMap(adjMap, falseColorsMap, cv::COLORMAP_AUTUMN);`
  - `cv::imshow("Out", falseColorsMap);`
- Suggerimento: limitatevi a visualizzare l'orientazione dove la magnitudo è superiore a una soglia prefissata

- Normalmente si considerano coordinate di pixel intere
  - Ma ci sono casi in cui ha senso considerarle a virgola mobile
  - Qual è il valore del pixel in questo caso?
- Realizzare una funzione di bilinear interpolation:
  - `float bilinear(const cv::Mat& src, float r, float c)`
    - `src`: singolo canale `uint8`
    - `r` e `c` valori float compresi in `[0,rows-1]` e `[0,cols-1]`
  - Il risultato sarà un singolo valore float ottenuto interpolando i 4 vicini di `(r,c)`

- Example

- $(r,c)=(27.8, 11.4)$

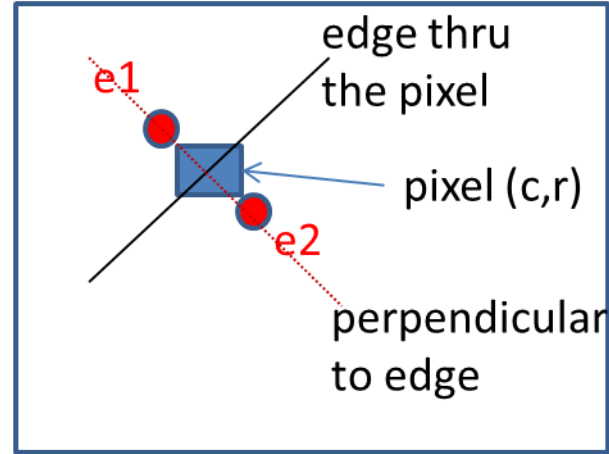
- $t=0.8, s=0.4$



- $$\text{out}(r,c) = \text{in}(27,11) \cdot (1-s) \cdot (1-t) + \text{in}(27,12) \cdot s \cdot (1-t) + \text{in}(28,11) \cdot (1-s) \cdot t + \text{in}(28,12) \cdot t \cdot s$$



- Non Maxima Suppression
- Dati i risultati di Sobel applicati al risultato del filtro gaussiano calcolare i massimi del gradiente in direzione perpendicolare al gradiente stesso
  - `int findPeaks(const cv::Mat& magn, const cv::Mat& or, cv::Mat& out)`
    - `magn`: singolo canale float32
    - `or`: singolo canale float32
    - `out`: singolo canale float32
- Confrontare ogni pixel dell'immagine magnitudo con i vicini e1, e2 a distanza 1.0 lungo la direzione del gradiente. Utilizzare la funzione bilinear del passo precedente.
  - Sui pixel di bordo scegliete voi la politica
- Visualizzare il risultato



$$e1x = c + 1 * \cos(\theta);$$

$$e1y = r + 1 * \sin(\theta);$$

$$e2x = c - 1 * \cos(\theta);$$

$$e2y = r - 1 * \sin(\theta);$$

Example:  $r=5$ ,  $c=3$ ,  $\theta=135$  degrees

$\sin \theta = .7071$ ,  $\cos \theta = -.7071$

$e1 = (2.2929, 5.7071)$

$e2 = (3.7071, 4.2929)$

- Sviluppare una funzione di soglia con isteresi stile “Canny”:
  - `int doubleTh(const cv::Mat& magn, cv::Mat& out, float t1, float t2)`
    - magnitude: singolo canale float32
    - out: singolo canale uint8 binarizzato
    - t1 e t2: soglie
- Applicarla ai risultati del passo precedente e visualizzare l'output