Università di Parma

Corso di Laurea Magistrale in Ingegneria Informatica

Fondamenti di Visione Artificiale

a.a. 2019/19

PROVA PRATICA 08-01-2019

NOME:

COGNOME:

MATRICOLA:

WORKSTATION N°:

Non è consentito scambiarsi materiale via rete (ovviamente).

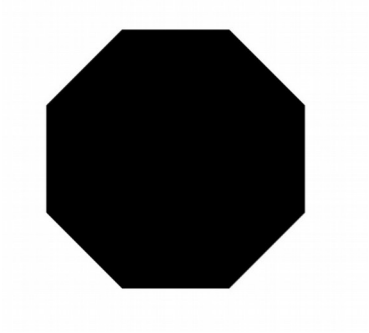
E’ consentito ’uso di funzioni OpenCv di alto livello come `at()` e similari.

Salvare l’esame in un file COGNOME_MATRICOLA.zip.

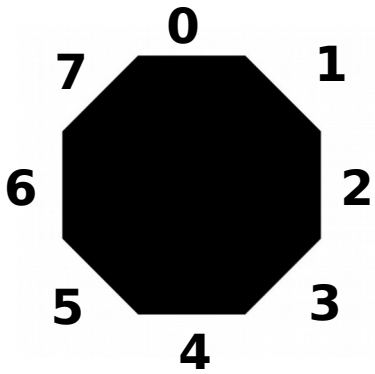
FIRMA

ES1

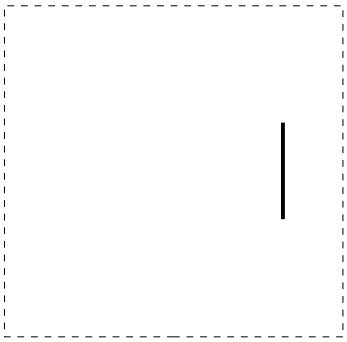
Data l’immagine “ottagono.pgm”:



Identifichiamo ogni lato dell’ottagono con un numero progressivo a partire da 0, in senso orario:



Scrivere un programma C/C++ che crei una **nuova** immagine contenente **unicamente il lato** corrispondente all’ultimio numero della matricola (eventualmente in modulo 8). Ad esempio, data una matricola 243532, questo è il risultato atteso:



Quindi *una singola linea*, in questo caso corrispondente *al lato 2*.

Esecuzione del codice di esempio:

./simple -i ../images/ottagono.pgm

HINTS:

1. Come sono le immagini gradiente orizzontale G_x e verticale G_y di "ottagono"? Provate di calcolarli.
2. Dati i gradienti orizzontali G_x e verticali G_y , quante sono le possibili combinazioni tra valori *positivi, negativi e nulli* (+,-,0) per ogni pixel? Posso sfruttare questa informazione?
3. E' un'immagine binaria perfetta, non e' necessario fare alcun tipo di noise reduction o smoothing, concentratevi sui gradienti.

ES2 (Extra)

In un caso generale, in cui la forma sia sempre convessa ma abbia piu' di 8 lati, come potrei risolvere questo problema? Come potrei individuare ogni orientazione nel caso generale?

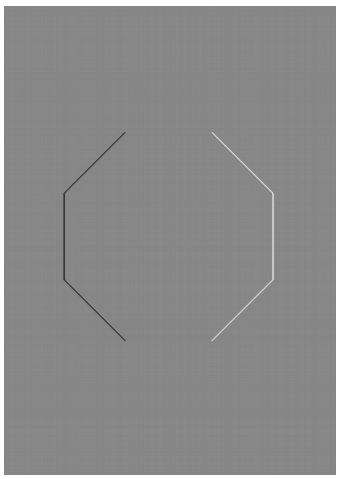
SOLUZIONE

ES1 ed ES2 sono di fatto lo stesso problema ma con due soluzioni diverse, una piu' semplice, l'altra piu' generale ed pulita.

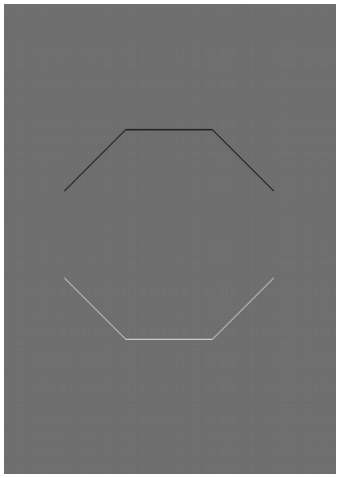
ES1

In ES1 suggerivo di analizzare i segni dei gradienti orizzontali e verticali in corrispondenza dei dati dell'ottagono:

Gx

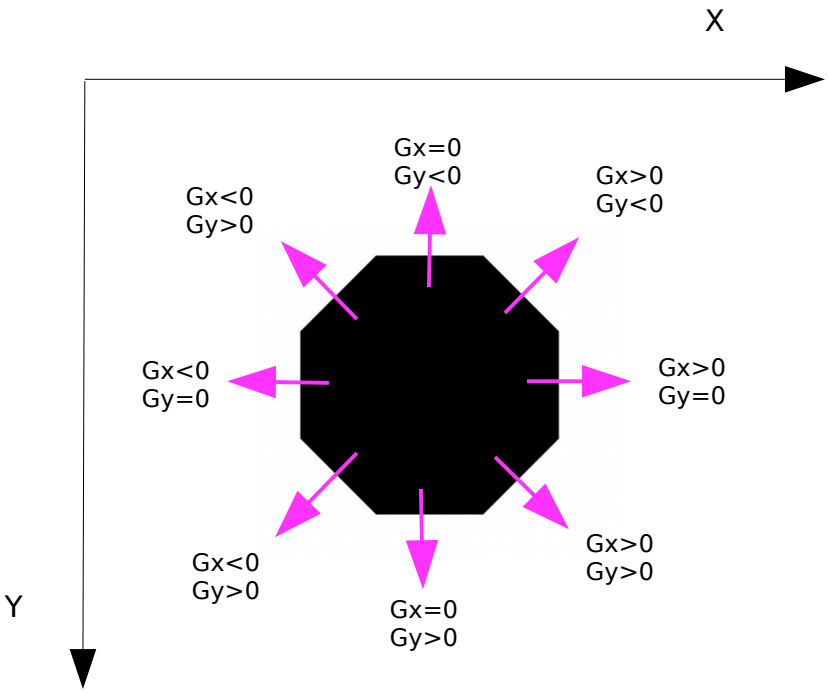


Gy



Dove il grigio corrisponde a gradiente 0, bianco gradiente massimo e positivo, nero gradienti minimo e negativo.

Esiste dunque una combinazione di segni di Gx e Gy che indentificano univocamente ogni lato:



Un possibile soluzione e’ dunque la seguente:

- calcolare i gradienti Gx e Gy con **SEGNO**
- In funzione del parametro di input “numero” individuare il lato cercato e disegnarlo; in altre parole mettere a 0 i pixel per i quali vale la condizione sopra indicata, corrispondente a “numero”

Il codice quindi doveva quindi contenere tutte le combinazioni di gradienti, e selezionare quello giusto in base al valore di “numero”.

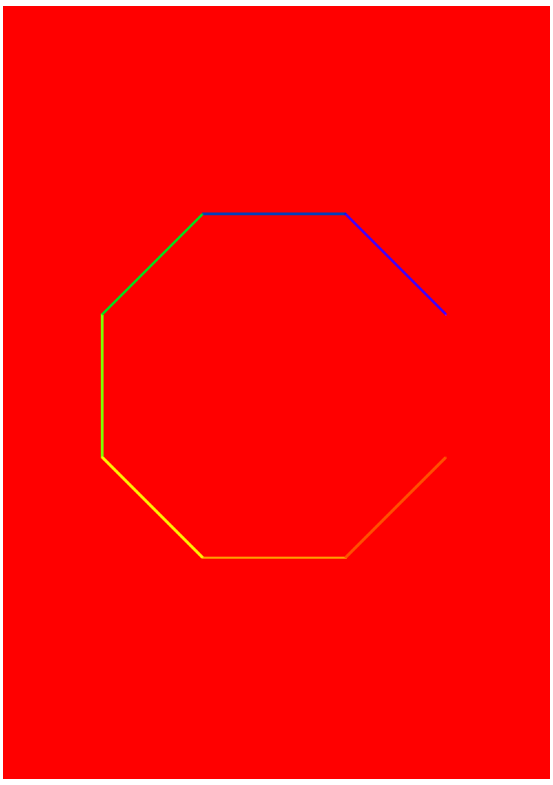
ES2

La soluzione precedente non e’ molto elegante e nemmeno generale: funziona solo per poligoni di massimo 8 lati. Se il numero di lati aumenta avremo, infatti, combinazioni di segni duplicate.

Il metodo piu’ generale e’ calcolare **l’ORIENTAZIONE** del gradienti, che, in caso di figure *convesse*, identifica in modo univoco un numero genericamente grade di lati.

Possibile soluzione:

- calcolare i gradienti Gx e Gy con **SEGNO**
- calcolare l’orientazione del gradiente come *arcotangente* di Gy/Gx. Nella figura sotto ogni colore corrisponde ad una diversa orientazione:



- in funzione del parametro di input “numero” individuare il lato e disegnarlo. Nel caso di figure equilatere, la relazione tra “numero” e angolo del gradiente puo’ essere espressa in questo modo:

2 * M_PI * numero / (Numero di lati del poligono)

Purtroppo nel testo ho fatto un’infelice scelta nella numerazione dei lati, per cui il lato a cui corrisponde un’arcotangente 0 e’ il numero 2... Per ottenere risultati esattamente uguali tra i due metodi e’ necessario modificare “numero” aggiungendo 6. Questo dettaglio implementativo ovviamente non era richiesto e verra’ ignorato.

NOTA IMPLEMENTATIVA GENERALE

Non e’ mai buona norma scrivere una condizione if/while/ecc. in cui si eguaglia un dato floating point ad un particolare valore, cercando una corrispondenza *esatta*.

Ad esempio, invece di questa condizione:

```
variabile_float == 0.0
```

si preferisce utilizzare questa:

```
|variabile_float| < epsilon
```

Con epsilon opportunamente piccolo.

In generale, quando voglio cercare uno specifico valore in floating point, si procede in questo modo:

```
|variabile_float – valore_cercato| < epsilon
```

Questo per problemi di approssimazione numerica. Il dato floating point ha una capacita’ limitata di apprezzare frazioni di interi, essendo un dato di dimensione finita (32 o 64 bit in genere). Anche se la sequenza di calcoli con cui e’ stato generato il valore dentro variabile_float e’ corretta dal punto di vista matematico, il risultato finale potrebbe essere solo *un’approssimazione* del valore atteso. Magari molto buona (es. errore di 10e-6), ma non esatta, per cui il controllo if/while *fallirebbe*.

Nell’ES1 i valori di partenza erano tutti interi ed erano coinvolte solo operazioni semplici, come somme e sottrazioni, per cui i risultati erano sempre esatti.

Gia’ in ES2 non e’ consigliabile andare a testare un’uguagliaza tra il risultato di atan2 e un particolare valore atteso di angolo in modo esatto.