Képfeldolgozás a gyakorlatban

Vágás, küszöbölés

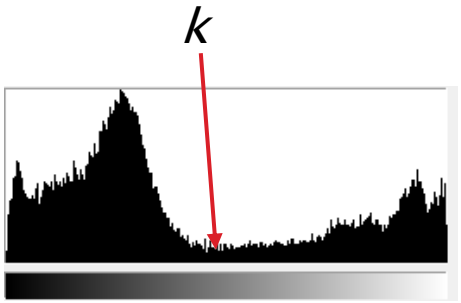
Szeghalmy Szilvia

Bináris küszöbölés

- Tegyük fel, hogy az objektum világosabb, mint a háttér (*vagy fordítva*)
- Keressük azt az intenzitásértéket, *küszöböt*, amelynél az előtérpontok világosabbak, a háttérpontok sötétebbek (*vagy fordítva*)
- A bináris küszöbölés *kétszintű* képet ad, vagyis a *k* küszöbnél kisebb egyenlő, illetve nagyobb értékekhez más-más színt rendelünk az eredményképen ($v_1 \neq v_2$):

$$I'(p) = \begin{cases} v_1 & , ha\ I(p) \leq k \\ v_2 & , különben \end{cases}$$

- Az eredménykép általában fekete-fehér:



Küszöbölés OpenCV-ben (globális)

```
threshold(InputArray src, OutputArray dest,
                double kuszob, double maxErtek, int eljaras );
```

`eljaras:`

- `THRESH_BINARY:` $dest(p) = \begin{cases} maxErtek & , ha\ src(p) > kuszob \\ 0 & , különben \end{cases}$
- `THRESH_BINARY_INV:` $dest(p) = \begin{cases} 0 & , ha\ src(p) > kuszob \\ maxErtek & , különben \end{cases}$
- `THRESH_OTSU:`
  - a küszöbértéket automatikus módon határozza meg
  - küszöb: az az érték, amelynél az előtérhez ill. a háttérhez sorolt pontok intenzitásértékeinek szórása (előtér_szórása + háttér_szórása) minimális.

Küszöbölés OpenCV-ben (globális)

- Töltse le az alábbi képet és olvassa be szürkeskálában:
  https://en.wikipedia.org/wiki/File:American_Eskimo_Dog.jpg

- Küszöböle valamelyik eljárással a szürkeskálás képet:
  - THRESH_BINARY eljárással, pl. 100-as küszöbbel
  - automatikus küszöbértékkel, a THRES_OTSU eljárással
  - és hasonlítóoperátor (<) segítségével

- Távolíts el a só jellegű zajokat (1 pixeles fehér értékek fekete környezetben) egy kisméretű mediánszűrővel.
- Jelenítse meg az eredményt

Vágás OpenCV-ben

- Vágás: A bináris küszöböléstől abban tér el, hogy a küszöb alá (vagy fölé) eső intenzitásértékeket meghagyjuk.

```
threshold(InputArray src, OutputArray dest,
                double kuszob, double maxErtek, int eljaras );
```

`eljaras`:

- `THRESH_TOZERO` : $dest(p) = \begin{cases} src(p) & , ha\ src(p) > kuszob \\ 0 & , különben \end{cases}$
- `THRESH_TRUNC` : $dest(p) = \begin{cases} maxValue & , ha\ src(p) > kuszob \\ src(p) & , különben \end{cases}$

- Feladat: Próbáljuk ki a THRESH_TOZERO-t az eszkimó kutyára!

Küszöbölés RGB képre

- Lehetőségek:
  a) Alakítsuk szürkeskálássá
  b) Nézzük meg, hogy melyik csatornát érdemes használni
     (csatornabontás: *split(const Mat src, vector<Mat>& dest_vect*))
  a) Keressünk alkalmasabb színteret.

szürkeskálás kép

R

G

B

http://fvaudubon.org/wp-content/gallery/whats-that-bird/dsc_1501-3s.jpg

Feladat

- Töltse be a madar.jpg képet színesben.
- Bontsa szét a csatornákat.
- Végezzen küszöbölést a B csatornán.
  A madár pontjai legyenek fehérek, a többi legyen fekete az eredményképen.
  (a tönk egy része a képen maradhat)
- Az eredeti képről másolja a madarat egy új fekete képre.
  - img.copyTo(C, maszk)
- Jelenítse meg az eredményt.

Sávkiemelés

- Sávkiemelés $0 < k_1 < k_2 < L$ küszöbértékek esetén $[0, L]$, intenzitástartományt feltételezve, ($v_1, v_2 \in [0, L]$, $v_1 \neq v_2$):

$$I'(p) = \begin{cases} v_2 & , ha\ k_1 \leq I(p) \leq k_2 \\ v_1 & , különben \end{cases}$$

- OpenCV-ben:

```
inRange(InputArray src, Scalar kuszob1,
          Scalar kuszob2, OutputArray dest);
```

  - Szürkeskálás és színes képen is működik
  - Egy csatornás, fekete-fehér képet ad (CV_8UC1)

Sávkivágás

- Megőrizzük az intenzitástartományon belüli értékeket.
- Sávki**vágás** $0 < k_1 < k_2 < L$ küszöbértékek esetén [0, L] intenzitástartományt feltételezve ($v \in [0, L]$):

$$I'(p) = \begin{cases} \boldsymbol{I(p)} & , ha\ k_1 \leq I(p) \leq k_2 \\ v & , különben \end{cases}$$

- Sávkizárás: (a sávon kívül őrzi meg az értékeket)

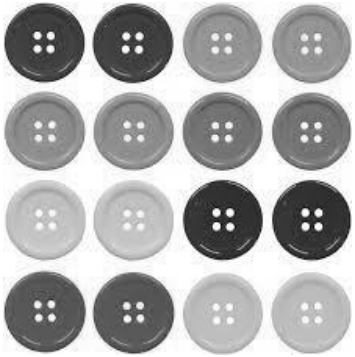
$$I'(p) = \begin{cases} \boldsymbol{v} & , ha\ k_1 \leq I(p) \leq k_2 \\ \boldsymbol{I(p)} & , különben \end{cases}$$

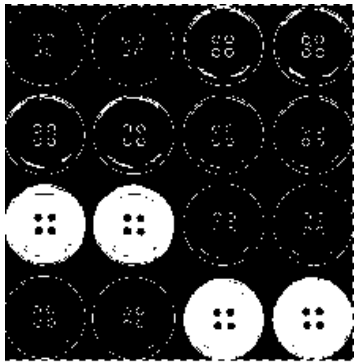
- Sávkivágás megvalósítása OpenCV-ben:
  - eredeti kép klónozása *(ha az erdetire még szükségünk van)*
  - sávkimelelés (inRange) -> megadja a sáv maszkját (bináris kép)
  - a maszk invertálása *(sávkizárásnál ez a lépés elmarad)*
  - a maszk alatti értékek 0-ra állítása a képen

    img.setTo(0, maszk)

Feladat: sávkivágás

- Olvasd be a gray_buttons képet
- A gombok kiemeléséhez használd a képen látott tartományt.
- Használj mediánszűrőt a zaj eltávolítására.
- Invertáld a maszkot.
- Állíts minden értéket nullára a maszk alatt.





Feladat: sávkizárás

- Töltse le a sas.avi videót.
  http://arato.inf.unideb.hu/szeghalmy.szilvia/kepfeld/img/sas.avi

- Videó olvasása:

```
VideoCapture cap;
cap.open(<fajlnev_eleresi_uttal>);
```

- Sikeres-e a megnyitás:

```
cap.isOpened()   //visszatérési érték: true, ha sikeres
```

- Egész videó olvasása:

```
Mat img;
while(1){          //végtlen ciklus
   cap >> img;
   if (img.empty()) break;
   képkocka feldolgozása
}
```

https://www.youtube.com/watch?v=thEkFMaB5CA

Feladat: sávkizárás

- Az ég intenzitásértékét tudjuk egy tartománnyal leírni.
  - A sas-hoz kettő kellene, ezért nem a sas pontjait próbáljuk sávkivágással megőrizni.
- Ezt a tartományt fogjuk kizárni (0 értékre állítani).

- A cikluson belül:
  - Klónozza az img képet
  - Alakítsa át a képet szürkeskálássá a *cvtColor* függvény segítségével
  - Az *inRange* függvénnyel segítségével határozza meg a **háttér** maszkját
    - alsó határ: 100
    - felső határ: 155
  - Mossa el a maszkot egy 5x5-ös mediánszűrővel.
  - A klón képen nullázza ki a maszk alá eső pontokat a setTo függvénnyel
  - Jelenítse meg/mentse el az eredeti és az eredményképet.
    mentés: imwrite(<elérési_út>, mentendő_kép);
  - Várakoztasson 33 ms-t.

Feladat: küszöbölés rögzített értékkel

- Készítsen egy programot, mely különböző küszöbértékek (20, 40, 60, ...) mellett küszöböli a képet:

  e-learning: sejtek.png

- Az eredmények összevetésénél előnyös, ha külön-külön ablakokban jelenítjük meg a küszöbölt képeket.

```
int x = 0, y = 0;
```

- Ciklus 20, 40, ... küszöbértékekre:
  - Hozzon létre egy ablakot. **Az ablaknévben szerepeljen a küszöbérték**.

    ```
    namedWindow(ablaknev, WINDOW_NORMAL);
    ```

  - Küszöbölje a képet és jelenítse meg az ablakban.
  - Méretezze át az ablakot (ne a képet) egy előre rögzített méretre (pl: 300x300).

    ```
    resizeWindow(ablaknev, ujmeret)
    ```

  - Mozgassa el úgy az ablakot, hogy ne kerüljön rá az előző ablakokra.

    ```
    x, y átállítása
    moveWindow(ablaknev, x, y)
    ```

- Várakoztatás billentyűzet leütésre.
- Az ablakok lezárása

  ```
  destroyAllWindows()
  ```

Küszöbölés előtérre vonatkozó feltételezés alapján

- Akkor jó, ha egy képsorozatnál az előtér-háttér arány közel állandó:
  pl. űrlapok, ábra nélküli könyvek
- Egy képre meghatározzuk, hogy az előtér a kép hány százaléka.
- Felhasználjuk az összes képnél.
- A brute-force megoldás: (img – eredeti kép, dest – célkép)

```
double n = img.rows*img.cols //pixelek száma
double fg_pixels = 0.1*n;    //előtérpontok száma
for(int th=0; th<256;++th){  //min 1, max 256 lépés, avg: 128
    threshold(img, th, 255, dest);          //n lépés (ciklus)
    if (countNonZero(dest) >= fg_pixels)    //n lépés (ciklus)
        break;
}
```

Legrosszabb esetben 256*2 = 512 alkalommal fogjuk bejárni a képet.

Küszöbölés előtérre vonatkozó feltételezés alapján

- Helyette:
  1) Számítsunk hisztogramot
  2) Kezdjük el összegezni a hisztogram elemeit (*t= 0, 1, 2,...,255* indexek)
     Ha az összeg meghaladja az előtérpontok számát, akkor mentsük a küszöbértéket (*t*). Ciklus vége.
  3) Küszöböljük a képet t-vel

- Lépések?
  1) n lépés (n a képpontok száma)
  2) minimum 1, maximum 256 lépés
     nincs beágyazott ciklus, nincs képbejárás, elenyésző a költség
  3) n lépés
- összesen 2*n* + 256 a legrosszabb esetre => 2x járjuk be képet és minimális további költségünk van

Hisztogramszámítás OpenCV-ben

```
calcHisto( InputArrayOfArrays kepek,
      const vector<int> csatornak,
      InputArray maszk,
      OutputArray hiszto,    #F -> float
      const vector<int> hiszto_meretek,
      const vector<float> hiszto_tartomanyok,//alsóhatár,felső határ,...
      bool akkumlalt
);
```

- kepek: egy vektorba (vector<Mat>) csomagolt kép vagy képek
- csatornak: melyik kép, mely csatornája legyen használva (a számozás folytonos, tehát ha az 1. kép 3 csatornás, akkor a második kép 0. csatornája a 3-as, 1. csatornája a 4-es stb.)
- maszk: egy fekete/fehér (CV_8UC1) kép, ahol fehér, azt veszi figyelembe a többi képen a hiszto. számítás során. (ha nincs rá szüksége, akkor: noArray())
- hiszto_meretek: pl. szürkeskálánál 256 különböző érték van, de előfordulhat, hogy nem akarunk ennyi szürkeségi szintet.
- hiszto_tartomanyok: pl. 8UC3-s RGB kép R, G, és B csatornájánál is [0, 255], de pl. 8UC3-s HSV kép H csatornája csak [0, 179]. Össze kell hangolni a hiszto_meretek-kel!
- akkumlalt: ha *true*-ra, akkor a képek között nem nullázza le a hisztogram tömböt.

Egy hisztogram készítő függvény

```
void createHisto(const Mat img, Mat& hiszto){
        //a hiszto az eredmeny, float típusú elemeket tartalmaz majd

        vector<Mat> kepek;
        kepek.push_back(img); // egy képet használunk

        vector<int> csatornak;
        csatornak.push_back(0); //a képnek a 0. csatornáját használjuk

        vector<int> hiszto_meretek;
        hiszto_meretek.push_back(256);  //szürkeárnyalatok száma

        vector<float> hiszto_tartomanyok;
        hiszto_tartomanyok.push_back(0.0f); //hol kezdődik a tartomány
        hiszto_tartomanyok.push_back(255.f);  //meddig tart

        //accumlate: marad false (nullázza a hisztogrammot)
        calcHist(kepek, csatornak, noArray(), hiszto, hiszto_meretek,
                                                hiszto_tartomanyok, false);
}
```

Feladat

- Írjon függvényt, mely az előtér százalék alapján meghatározza az ideális küszöbértéket:

```
int calc_th_value(const Mat src, float ratio = 0.1f);
```

  - Számolja ki az előtérpontok elvárt számát (n_fg)

  - Hozzon létre egy hisztogramot a világosságértékek alapján
  - Kezdje el a tömb elejétől összegezni a tömböt (1 D-s mátrix):

```
...
for(int i = 0; i<histo.rows; ++i)
     s += histo.at<float>(i); # az OpenCV calcHist esetén
     s += histo.at<int>(i); a "histo.h"-s Histo::calcHisto-nál
     ...
```

  - Ha az aktuális összeg meghaladja az n_fg értékét, adja vissza az i értéket (küszöb)
  - A függvény végén adjon vissza 255-ös értéket. (A felh. 1 feletti ratio-t adott…)

Feladat - tesztelése

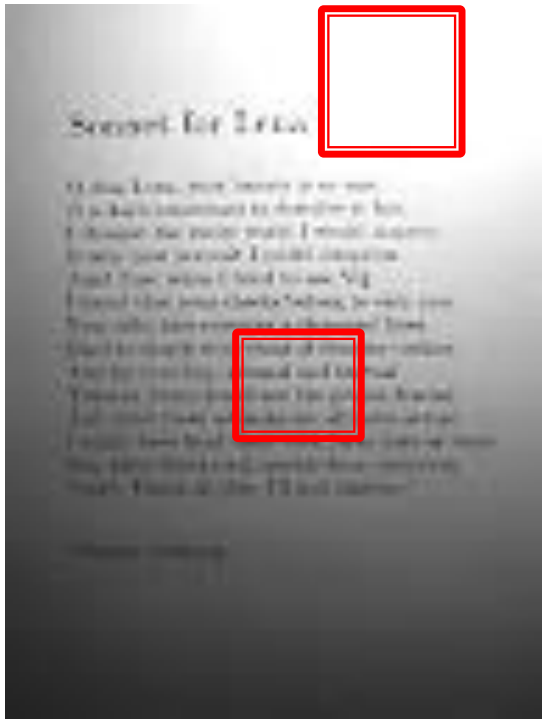
- Töltse be a scanned3.png-t szürkeskálában.
- Határozza meg a küszöbértéket. A feltételezésünk szerint a kép 10%-a előtér.

  (ratio = 0.1f)
- Küszöbölje a képet a kapott értékkel.
- Jelenítse meg az eredményt.

Adaptív küszöbölés, vágás

- Lokális környezetre értelmezett

```
adaptiveThreshold( InputArray src, OutputArray dest,
  double maxVal, int adaptivMethod, int thresholdMethod,
  int block_size, double C);
```



- thresholdMethod:

  THRESH_BINARY
  THRESH_BINARY_INV

- adaptiveMethod :

  ADAPTIVE_THRESH_MEAN_C

  (küszöb = a blokk alatti pontok intenzitásának átlaga – C)

  ADAPTIVE_THRESH_GAUSSIAN_C

  (küszöb = a blokk alatti pontok súlyozott átlaga - C)

Feladat: adaptív küszöbölés

- A feladat egy GI-s hallgató által fényképezett régi adatszerkezet zh küszöbölése.

  e-learning: zh.jpg

- Az eredményképen a szöveg fekete háttér előtt, fehérrel jelenjen meg.

- Az ablak méretét és az átlagértékből való levonást a felhasználó választhassa meg.

- Trackbar létrehozása (ha cikluson belül használja, akkor nem szükséges eseménykezelőt írni):

```
int sugar = 20;
...
createTrackbar("kuszob", <ablak_neve>, &sugar, 255);
```

Eseménykezelős változat

- Eseménykezelő:

```
void onChange_adthreshold(int pos, void* userdata){
  Mat src = *(cv::Mat*)userdata;
  Mat dest;
  //kuszobold a képet
  //jelenítsd meg az eredményt (dest)
}
```

- Trackbar hozzáadása a "zh" ablakhoz, összerendelés az eseménykezelővel:

```
Mat img = imread(<kép_elérési_út>, IMREAD_GRAYSCALE);
...
int sugar = 20;
createTrackbar("sugar", <ablak neve>, &sugar, 20, onChange_adthreshold,
&img);
...
waitKey(0);
```


99 little bugs in the code.
99 little bugs in the code.
Take one down, patch it around.
127 little bugs in the code...