Computer Vision

Catalin Stoean
catalin.stoean@inf.ucv.ro
http://inf.ucv.ro/~cstoean

Filtrarea imaginilor Obiective

- Filtrarea imaginilor folosind filtrare
  - Omogena
  - Gausiana
  - Mediana
  - Sobel – utilizat pentru detectare de margini
  - Laplace – utilizat pentru detectare de margini
  - Bilaterala

Filtrarea imaginilor

- Filtrarea are principalul rol de a elimina zgomotul din imagini.
  - Este utilizata si pentru a extrage anumite caracteristici vizuale
- Este o metoda obisnuita de preprocesare a imaginilor in vederea imbunatatirii rezultatelor finale.
- Termenii utilizati in engleza pentru acest gen de procesare:
  - Smoothing
  - Blurring

Filtrarea imaginilor

- Filtrarea se obtine aplicand un kernel (filtru) asupra unei imagini prin transformarea valorii pentru fiecare pixel intr-o valoare bazata pe kernel si pe pixelii din vecinatate in imaginea originala.
- Matematic, spunem ca are loc o convolutie intre imagine si kernel.

Exemple kernele

Cel mai simplu: box-filter normalizat

1/5

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

Cel mai folosit: Gaussian

1/273

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

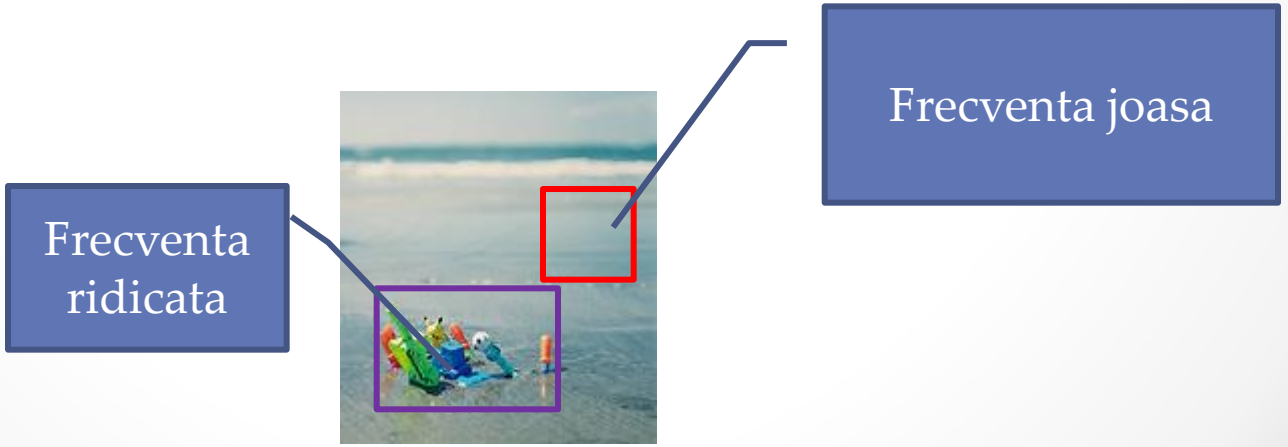
- Numarul de linii si coloane trebuie sa fie impar (3x3, 7x7, 11x17, etc)
- Cand marimea kernelui creste, timpul de procesare creste.

Filtrarea imaginilor

- Imaginile pot fi analizate dupa
  - Domeniul spatial: observarea distributiei nivelului de gri (sau a culorilor)
  - Domeniul frecventelor: variatiile din imagini
    - Unele au parti cu intensitati constante (de exemplu un cer albastru), in altele apar variatii rapide (ex: numeroase obiecte mici).

Domeniul frecventelor

- Imaginea este descompusa de la cele mai joase la cele mai ridicate frecvente.
- Frecventele joase corespund regiunilor unde intensitatile variaza usor
- Frecvente ridicate – in regiuni unde sunt schimbari bruste in intensitati.



Filtre pentru frecventa joasa

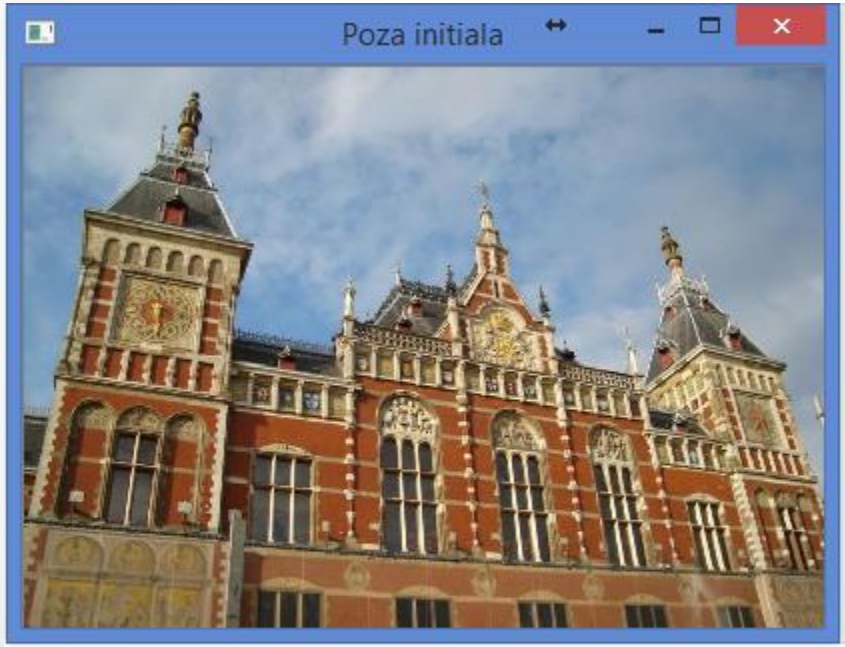
- O modalitate simpla de a reduce amplitudinea variatiilor este de a inlocui fiecare pixel cu valoarea medie a pixelilor din jur.
- Marimea kernelului afecteaza direct proportional cat de mari sunt modificarile in imagine.

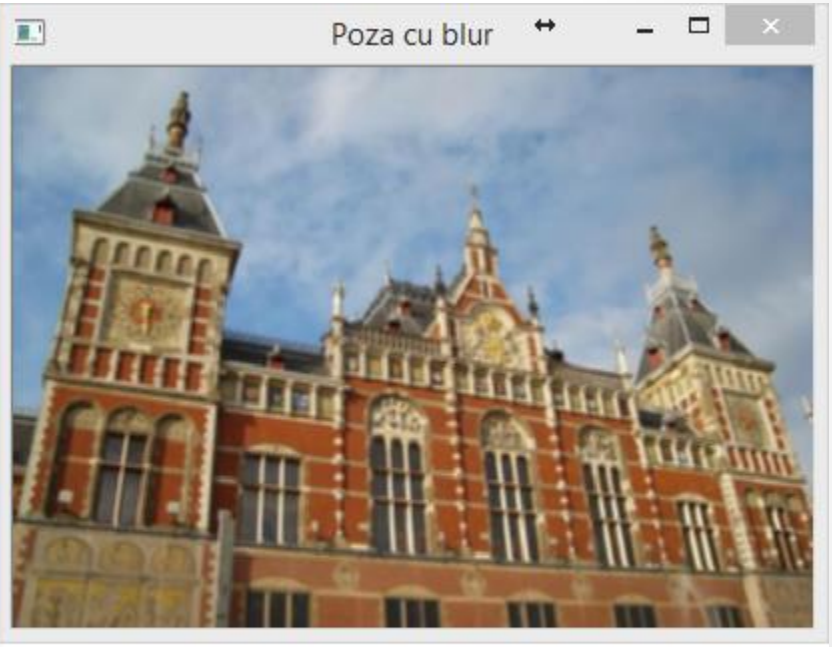
```
blur(poza, rezBlur, Size(5,5));
```

- Se citeste poza, se afiseaza in fereastra, dupa care se afiseaza in fereastra rezBlur.

blur

- Foloseste box filter normalizat.






```
blur(poza, rezBlur, Size(5,5));

namedWindow("Poza cu blur", WINDOW_NORMAL);
resizeWindow("Poza cu blur", 400, 280);
imshow("Poza cu blur", rezBlur);
```

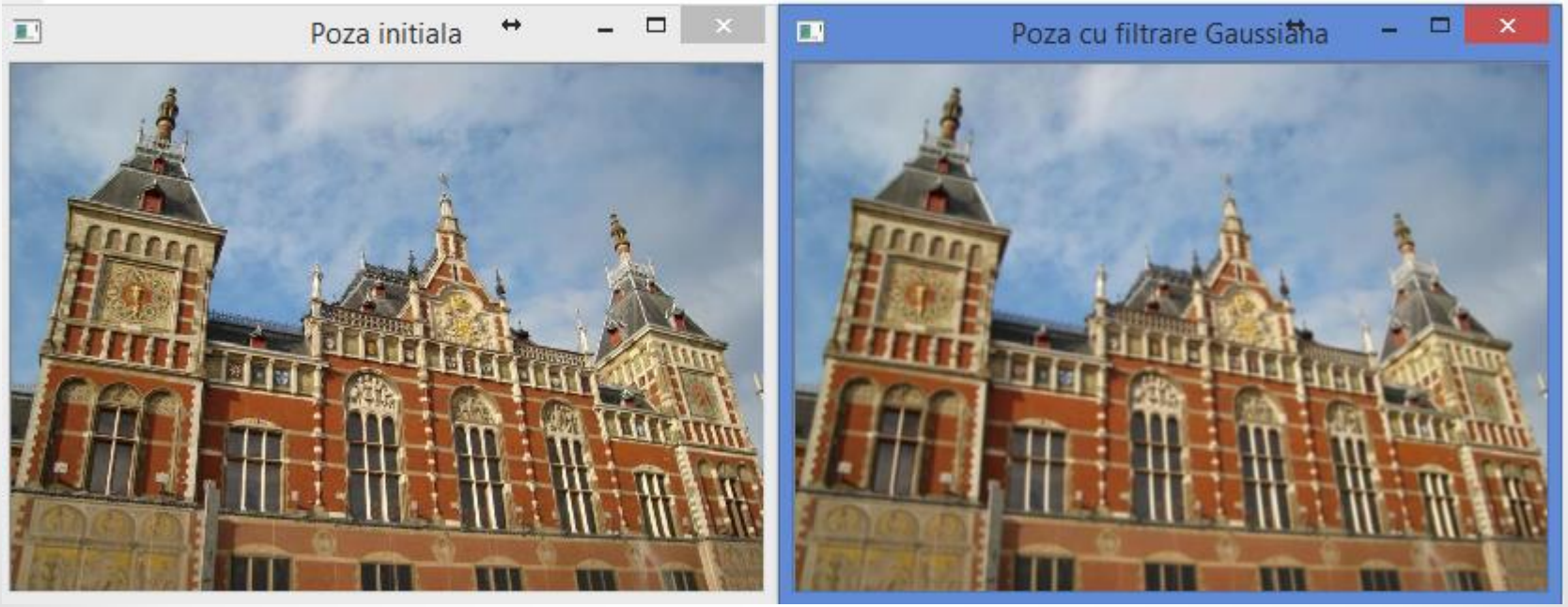
blur

- **void blur(InputArray src, OutputArray dst, Size ksize, Point anchor=Point(-1,-1), int borderType = BORDER_DEFAULT)**
  - **src** - imaginea sursa. (depth pentru imagine este unul dintre urmatoarele: CV_8U, CV_16S, CV_16U, CV_32F or CV_64F)
  - **dst** - rezultatul
  - **ksize** - marimea kernel-ului (valori impare, pozitive)
  - **anchor** - Point(-1,-1) inseamna ca ancora este in mijlocul kernel-ului
  - **borderType** - afecteaza pixelii din margine.
    - optiuni; BORDER_DEFAULT, BORDER_REFLECT, BORDER_REPLICATE, BORDER_TRANSPARENT, BORDER_REFLECT_101

Filtrare Gaussiana

- Pixelii care sunt mai apropiati pot fi considerati sa influenteze mai mult valoarea pixelului curent.

```
GaussianBlur(poza, rezGauss, Size(5,5), 1.5);

namedWindow("Poza cu filtrare Gaussiana", WINDOW_NORMAL);
resizeWindow("Poza cu filtrare Gaussiana", 400, 280);
imshow("Poza cu filtrare Gaussiana", rezGauss);
```



Filtrare Gaussiana

- **void GaussianBlur(InputArray src, OutputArray dst, Size ksize, double sigmaX, double sigmaY=0, int borderType=BORDER_DEFAULT)**
  - **src** si **dst**- imaginile sursa si destinatie
  - **ksize** - marimea kernel-ului
  - **sigmaX** - deviatia standard in directia lui X. Daca se foloseste 0, se calculeaza automat din ksize.
  - **sigmaY** - Similar cu sigmaX, dar in directia Y.
  - **borderType** - afecteaza pixelii din margine.
    - optiuni; BORDER_DEFAULT, BORDER_REFLECT, BORDER_REPLICATE, BORDER_TRANSPARENT, BORDER_REFLECT_101

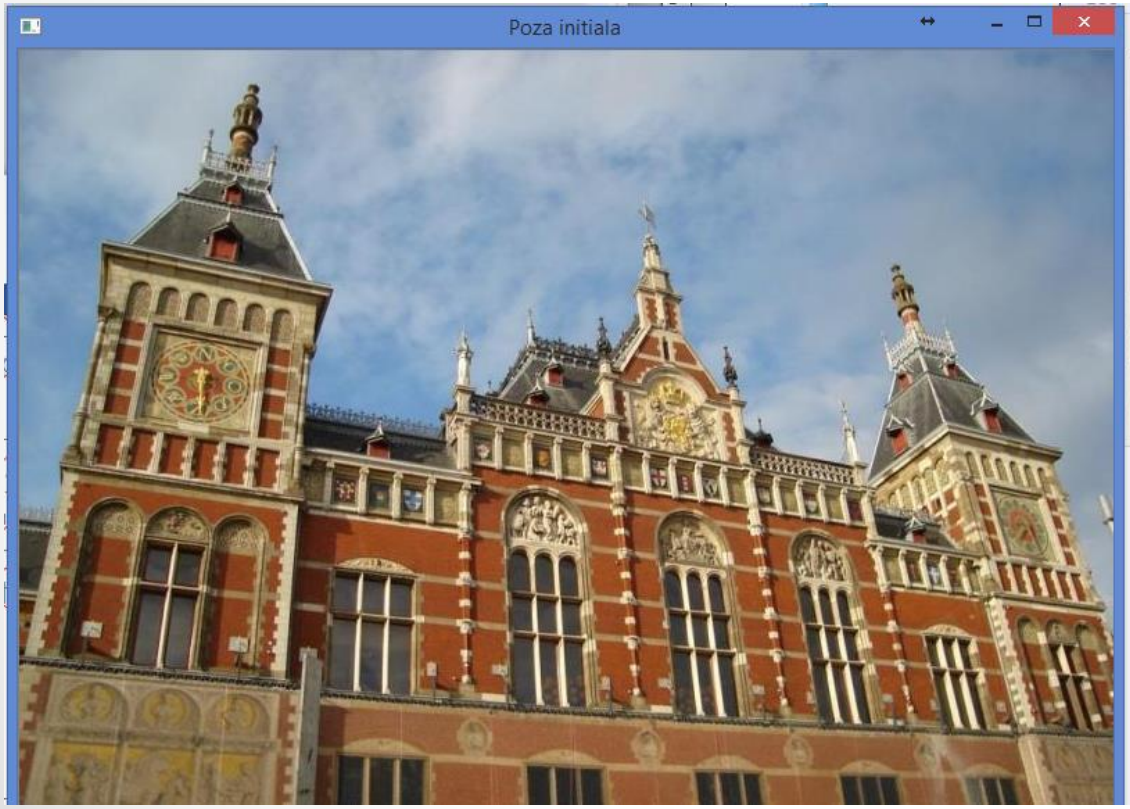
Redimensionare

- Redimensionarea pozelor foloseste filtre pentru frecventa joasa.
- O idee de redimensionare: eliminam liniile si coloanele din 2 in 2
  - Imaginea ar fi deteriorata
- Daca insa aplicam filtrare si apoi eliminam astfel linii si coloane, rezultatul este bun.

```
pyrDown(poza, pozaRedusa);
```

- Acest lucru este realizat de metoda de mai sus.

Redimensionare




```
{
    Mat pozaRedusa;

    pyrDown(poza, pozaRedusa);

    namedWindow("Poza redusa", WINDOW_AUTOSIZE);
    imshow("Poza redusa", pozaRedusa);
```

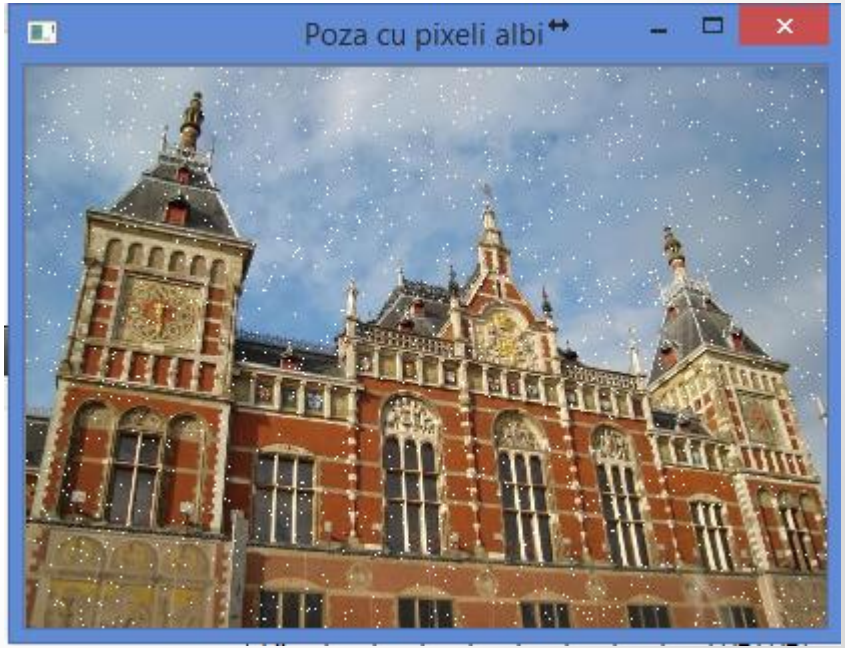
Redimensionare

- Metoda pyrDown foloseste un filtru Gaussian pentru imagine.
- Exista si **pyrUp**.
- De asemenea, exista si functia mai generala resize care ne permite sa specificam marimea imaginii rezultat (mai mare sau mai mica).

```
Mat pozaRedusa2;
resize(poza, pozaRedusa2, Size(poza.cols/3, poza.rows/3));
```

Filtrul median

- Este neliniar.
- Se foloseste in cadrul algoritmilor pentru detectarea marginilor pentru ca pastreaza relativ bine marginile si elimina zgomotul.
- Este util pentru eliminarea zgomotului, cum ar fi pixelii albi introdusi in prezentarea a doua.
-

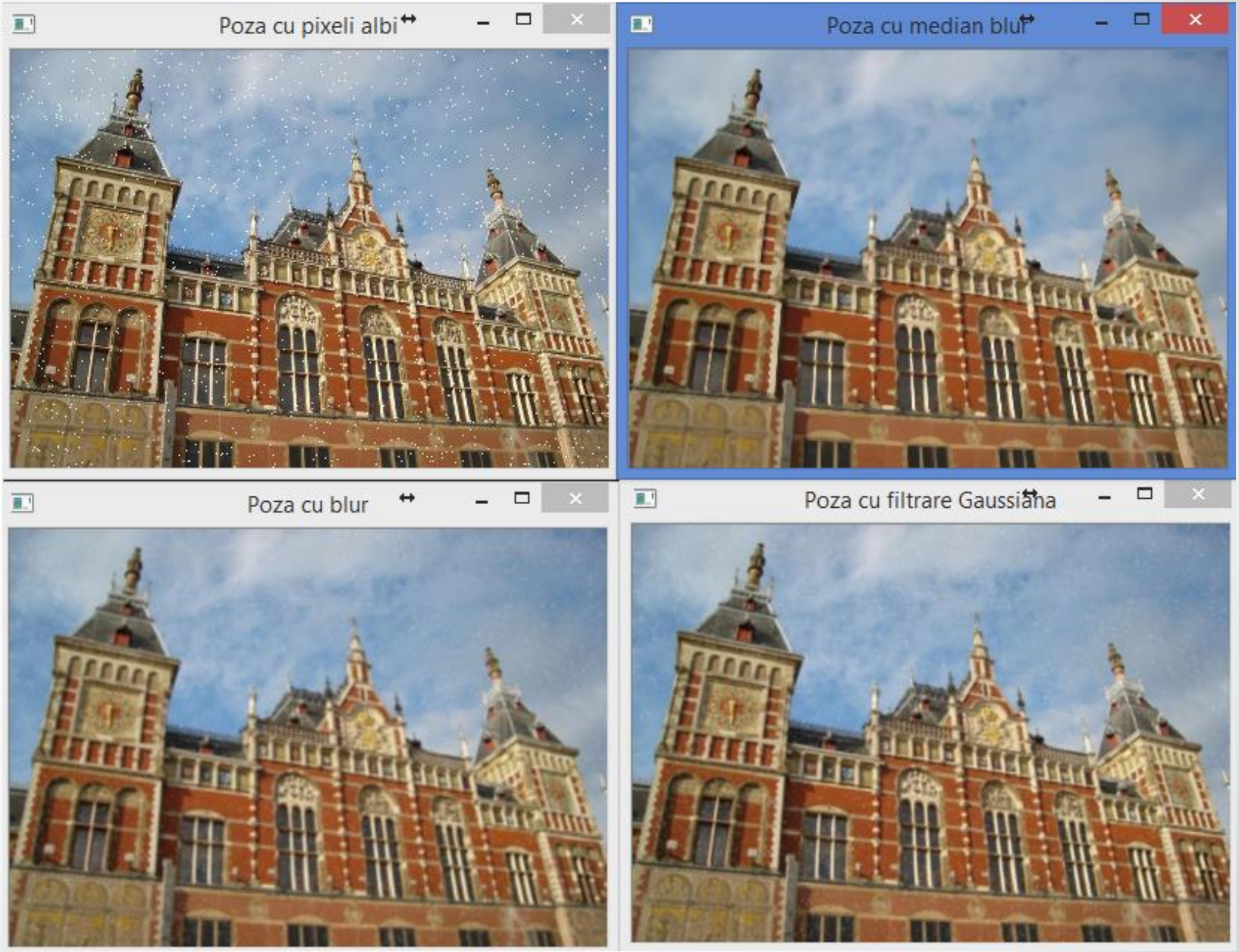


Filtrul median

```
medianBlur(poza, rezMedianBlur, 5);

namedWindow("Poza cu median blur", WINDOW_NORMAL);
resizeWindow("Poza cu median blur", 400, 280);
imshow("Poza cu median blur", rezMedianBlur);
```

- **void medianBlur(InputArray src, OutputArray dst, int ksize)**
  - **src** - imaginea de intrare.
    - Depth poate fi CV_8U pentru orice valoare a "**ksize**".
    - Pentru **ksize** = 3 sau **ksize** = 5, depth poate fi si CV_16U si CV_32F.
  - **dst** - imaginea de iesire
  - **ksize** - marimea filtrului (impar si mai mare decat 1)

Median blur are zgomotul eliminat.



Filtrul median

- Opereaza asupra pixelilor din vecinatate pentru a determina valoarea pixelului curent.
- Se calculeaza mediana pixelilor din vecinatate (inclusiv cel curent) si aceasta inlocuieste valoarea curenta
  - Acesta este motivul pentru care functioneaza atat de bine la eliminarea zgomotului

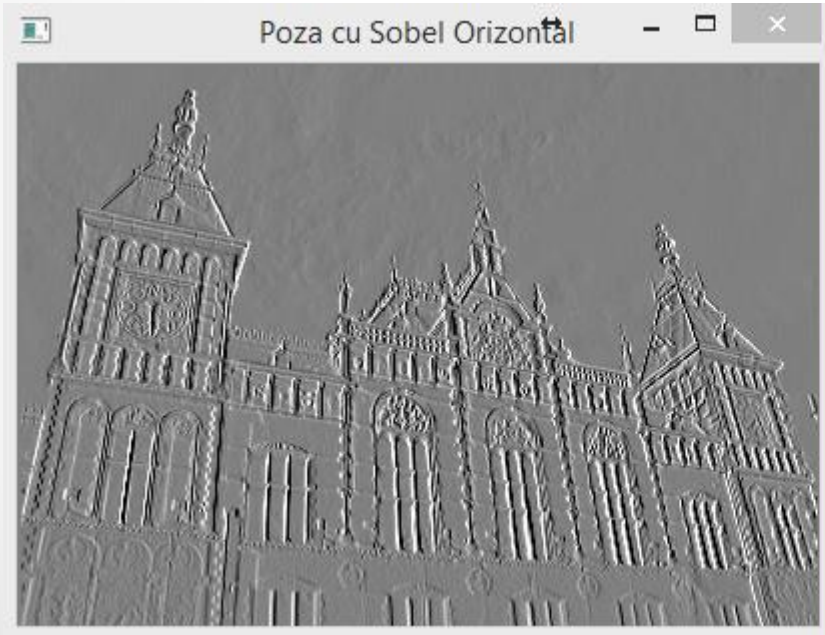
Filtre directionale - Sobel

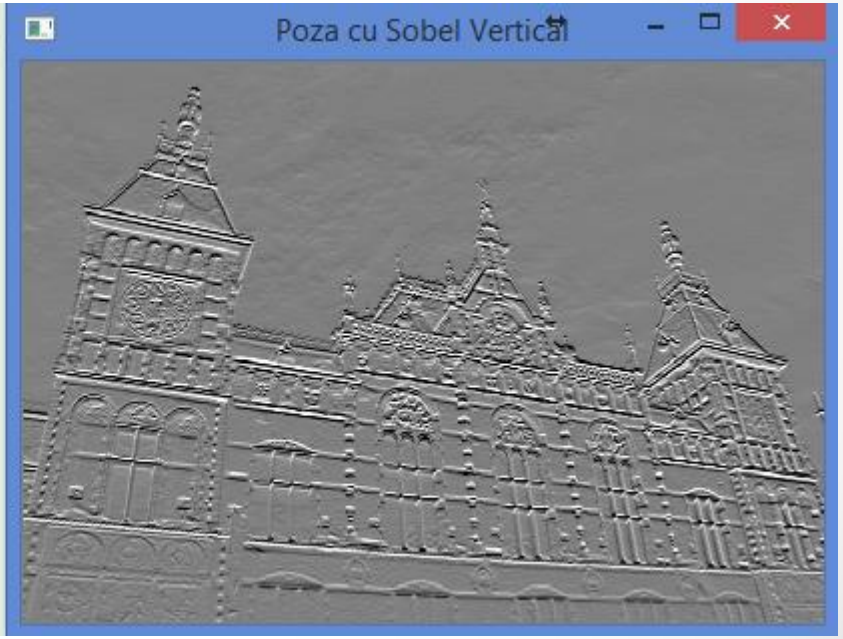
- Este un filtru pentru frecvente ridicate
- Se aplica doar asupra frecventelor aflate pe orizontala sau verticala unei imagini, in functie de kernelul selectat
- Da o aproximare a gradientului imaginii prin diferentierea pixelilor pe orizontala sau verticala

```
Sobel(poza, rezSobelOriz, CV_8U, 1, 0, 3, 0.4, 128);
namedWindow("Poza cu Sobel Orizontal", WINDOW_NORMAL);
resizeWindow("Poza cu Sobel Orizontal", 400, 280);
imshow("Poza cu Sobel Orizontal", rezSobelOriz);

Sobel(poza, rezSobelVert, CV_8U, 0, 1, 3, 0.4, 128);
namedWindow("Poza cu Sobel Vertical", WINDOW_NORMAL);
resizeWindow("Poza cu Sobel Vertical", 400, 280);
imshow("Poza cu Sobel Vertical", rezSobelVert);
```

Filtrul Sobel

- Se realizeaza o imagine pe 8 biti, unsigned (CV_8U).
- Seamana cu efectul *emboss*.





Sobel

- void Sobel(InputArray src, OutputArray dst, int ddepth, int dx, int dy, int ksize=3, double scale=1, double delta=0, int borderType=BORDER_DEFAULT)
  - src si dst = imaginile de intrare si iesire
  - ddepth = tipul de imagine (CV_8U, CV_16S, CV_32F, CV_64F)
  - dx, dy = specificatii ale kernelului cu privire la ordinul derivatei pe orizontala/verticala
  - ksize = marimea kernelului (1, 3, 5 sau 7)
  - scale = factor de scalare pentru derivate. Implicit, este fara scalare
  - delta = valoare care se adauga la rezultat in dst
  - **borderType** - afecteaza pixelii din margine.
    - optiuni; BORDER_DEFAULT, BORDER_REFLECT, BORDER_REPLICATE, BORDER_TRANSPARENT, BORDER_REFLECT_101

Filtrul Sobel

- Filtrele care calculeaza derivate pentru imagini sunt pentru frecvente ridicate
- In exemplul nostru, ultimul parametru 128 face sa fie imaginile pe un fundal gri
  - Daca era 0, ar fi fost negru
- Orizontal (parametrii 4 si 5): 1,0
- Vertical: 0,1

```
Sobel(poza, rezSobelOriz, CV_8U, 1, 0, 3, 0.4, 128);
namedWindow("Poza cu Sobel Orizontal", WINDOW_NORMAL);
resizeWindow("Poza cu Sobel Orizontal", 400, 280);
imshow("Poza cu Sobel Orizontal", rezSobelOriz);

Sobel(poza, rezSobelVert, CV_8U, 0, 1, 3, 0.4, 128);
namedWindow("Poza cu Sobel Vertical", WINDOW_NORMAL);
resizeWindow("Poza cu Sobel Vertical", 400, 280);
imshow("Poza cu Sobel Vertical", rezSobelVert);
```

Filtrul Sobel

- Kernelul sau contine atat valori pozitive, cat si negative, prin urmare rezultatul de la Sobel este adesea pus calculat in CV_16S (signed integer).
- Cele doua rezultate (verticala si orizontala) se combina apoi pentru a obtine norma filtrului Sobel

```
Sobel(poza, rezSobelOriz, CV_16S, 1, 0);
Sobel(poza, rezSobelVert, CV_16S, 0, 1);
//obtinem mai departe norma filtrarii Sobel
normaSobel = abs(rezSobelOriz) + abs(rezSobelVert);
```

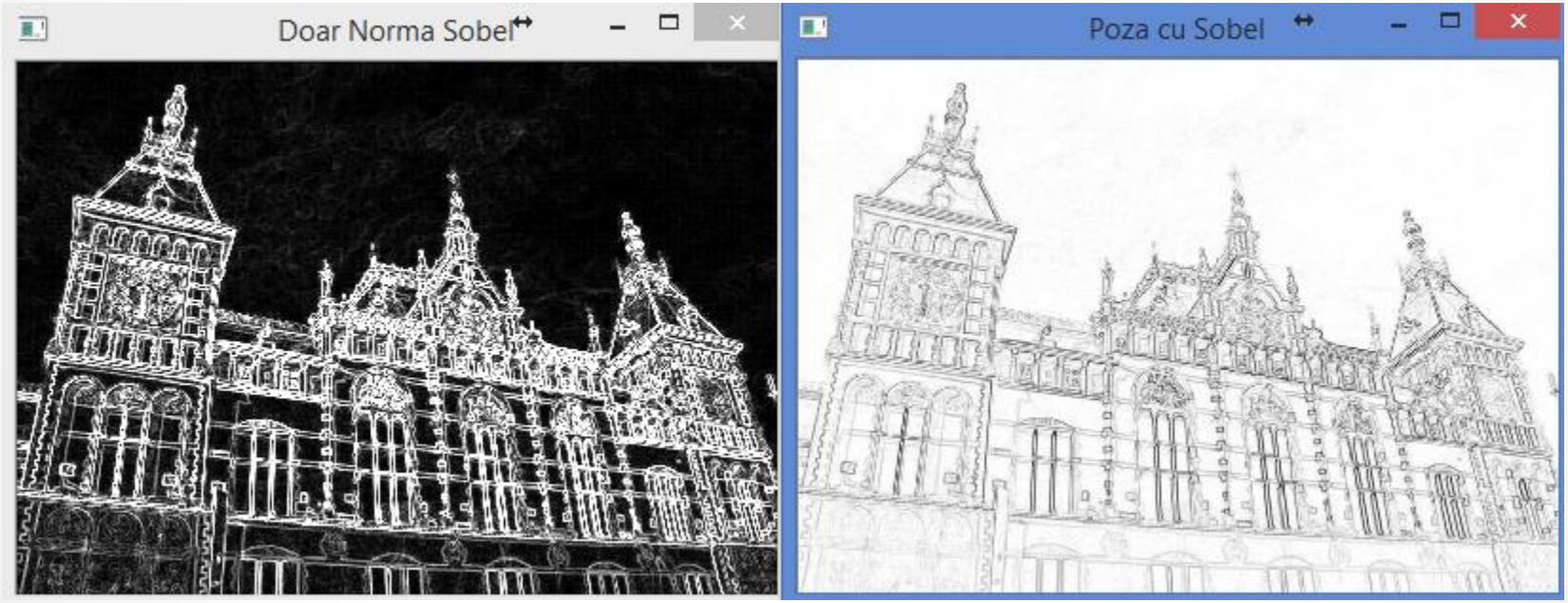
Filtrul Sobel

- Pentru a putea vedea norma Sobel, trebuie sa o convertim la CV_8U.
- Mai mult, putem face o convertire in care valorile de 0 corespund la alb, iar cele mai mari sa corespunda la nunte de gri.

```
normaSobel.convertTo(doarNormaSobel, CV_8U);

namedWindow("Doar Norma Sobel", WINDOW_NORMAL);
resizeWindow("Doar Norma Sobel", 400, 280);
imshow("Doar Norma Sobel", doarNormaSobel);

//gasim valorile de min si max din norma
double sobMin, sobMax;

minMaxLoc(normaSobel, &sobMin, &sobMax);

//convertim la o imagine 8-bit unde 0 corespunde la alb
normaSobel.convertTo(rezSobel, CV_8U, -255./sobMax, 255);

namedWindow("Poza cu Sobel", WINDOW_NORMAL);
resizeWindow("Poza cu Sobel", 400, 280);
imshow("Poza cu Sobel", rezSobel);
```

in.convertTo (out, depth, scale, shift)
out(i) = in(i) * scale + shift

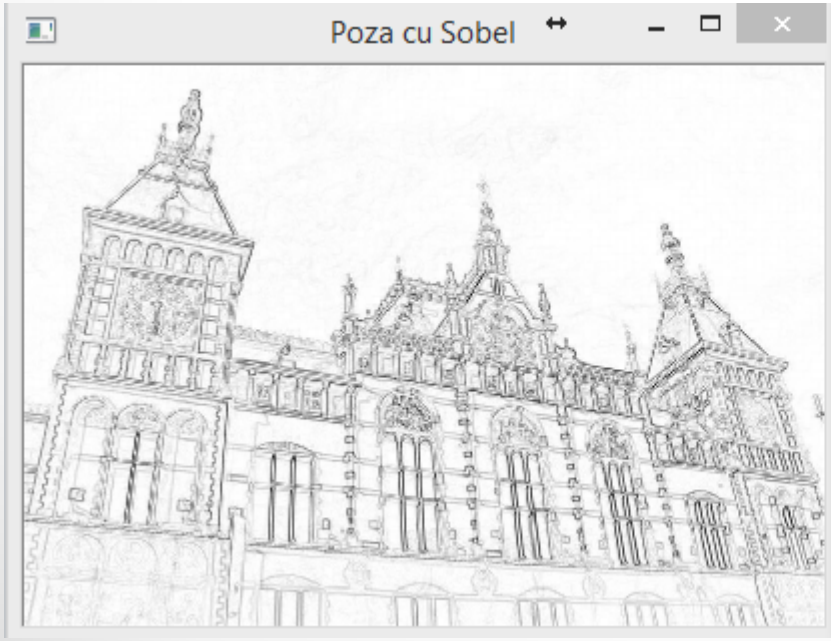
Filtrul Sobel



Filtrul Sobel

- Pentru a fi mai clar vizibile marginile din poza, se poate aplica un thresholding in care contururile sa fie desenate cu negru.

```
//incercati diverse valori pentru prag - aici 220
threshold(rezSobel, sobelPrag, 220, 255, THRESH_BINARY);
namedWindow("Poza cu Sobel prag", WINDOW_NORMAL);
resizeWindow("Poza cu Sobel prag", 400, 280);
imshow("Poza cu Sobel prag", sobelPrag);
```

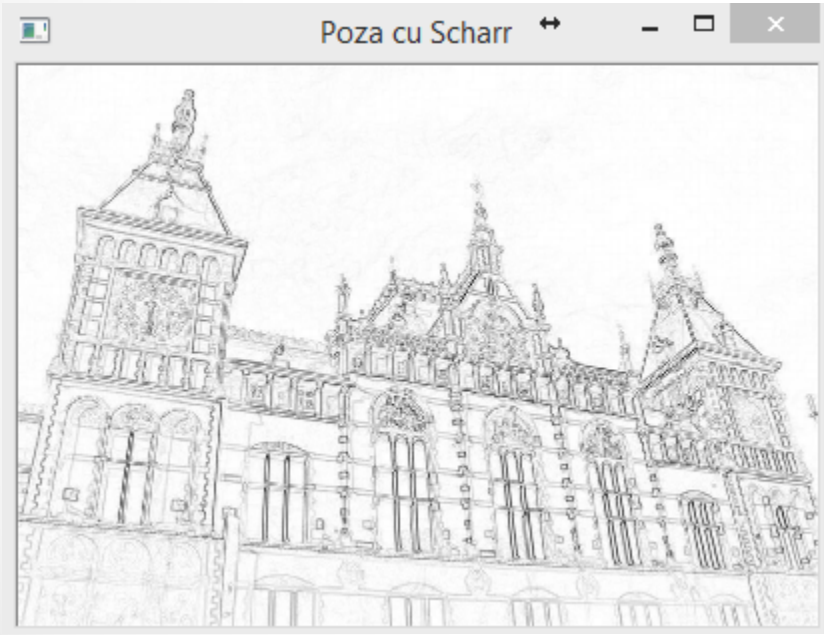


Filtrul Scharr

- Alt filtru care se bazeaza tot pe gradient si da estimari mai precise este Scharr.

```
Sobel(poza, rezScharrOriz, CV_16S, 1, 0, CV_SCHARR);
Sobel(poza, rezScharrVert, CV_16S, 0, 1, CV_SCHARR);
//obtinem mai departe norma filtrarii Scharr
normaScharr = abs(rezScharrOriz) + abs(rezScharrVert);
//gasim valorile de min si max din norma
double scharrMin, scharrMax;

minMaxLoc(normaScharr, &scharrMin, &scharrMax);

//convertim la o imagine 8-bit unde 0 corespunde la alb
normaScharr.convertTo(rezScharr, CV_8U, -255./scharrMax, 255);

namedWindow("Poza cu Scharr", WINDOW_NORMAL);
resizeWindow("Poza cu Scharr", 400, 280);
imshow("Poza cu Scharr", rezScharr);

//incercati diverse valori pentru prag - aici 220
threshold(rezScharr, scharrPrag, 220, 255, THRESH_BINARY);
namedWindow("Poza cu Scharr prag", WINDOW_NORMAL);
resizeWindow("Poza cu Scharr prag", 400, 280);
imshow("Poza cu Scharr prag", scharrPrag);
```

Filtrul Scharr



Filtrul Laplace

- Este un filtru pentru frecvente ridicate
- Calculeaza derivate de ordinul II.
- Cream o clasa dedicata.

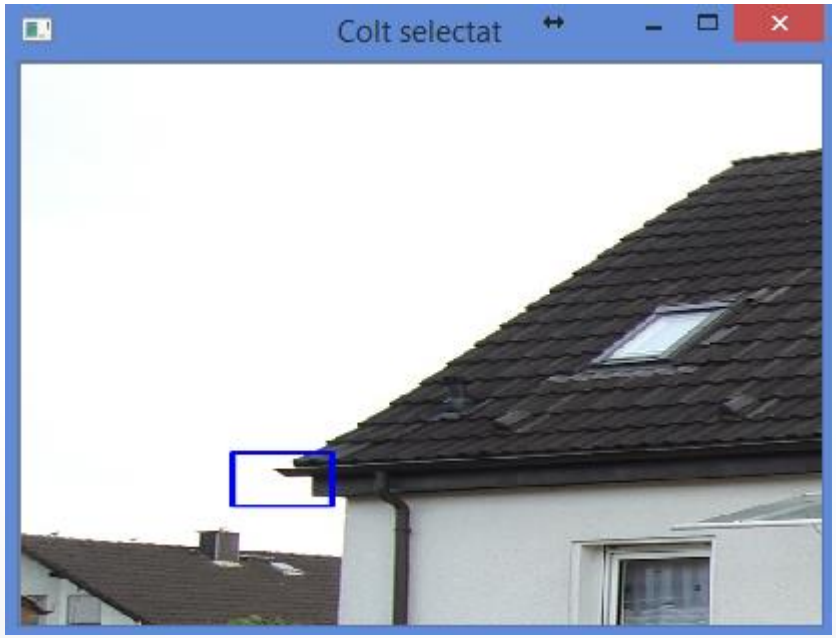
Filtrul Laplace

- Calculul pentru Laplacian este facut pentru o imagine CV_32F.
- Pentru a vedea imaginea, realizam o scalare
  - 0 este schimbat in 128

```
//modificam marimea aperturii kernelului
void setAperture(int a)
{
    aperture = a;
}

//calculam Laplacianul CV_32F
Mat computeLaplacian(const Mat& image)
{
    Laplacian(image, laplace, CV_32F, aperture);
    //pastram o copie locala a imaginii pentru
    //a desena contururi
    img = image.clone();
    return laplace;
}

/* Extragem rezultatul Laplace intr-o imagine pe 8 biti
 * 0 corespunde nivelului de gri 128
 * Daca nu se foloseste scalare, valoarea max va fi 255
 * computeLaplacian trebuie apelat inaintea acestei metode
 */
Mat getLaplacianImage(double scale=-1.0)
{
    if (scale<0)
    {
        double lapmin, lapmax;
        minMaxLoc(laplace,&lapmin,&lapmax);
        scale= 127/ max(-lapmin,lapmax);
    }
    Mat laplaceImage;
    laplace.convertTo(laplaceImage,CV_8U,scale,128);
    return laplaceImage;
}
```

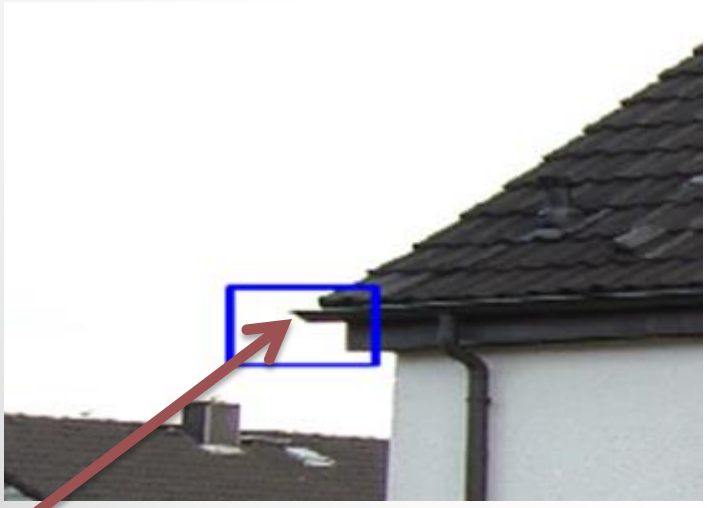
Filtrul Laplace

- Este sensibil la zgomot
- Pentru o mai buna exemplificare de extragere de contururi, schimbam imaginea cu una care are si sectiuni cu variatii mici.
- Coltul selectat va fi studiat in special.



Filtrul Laplace

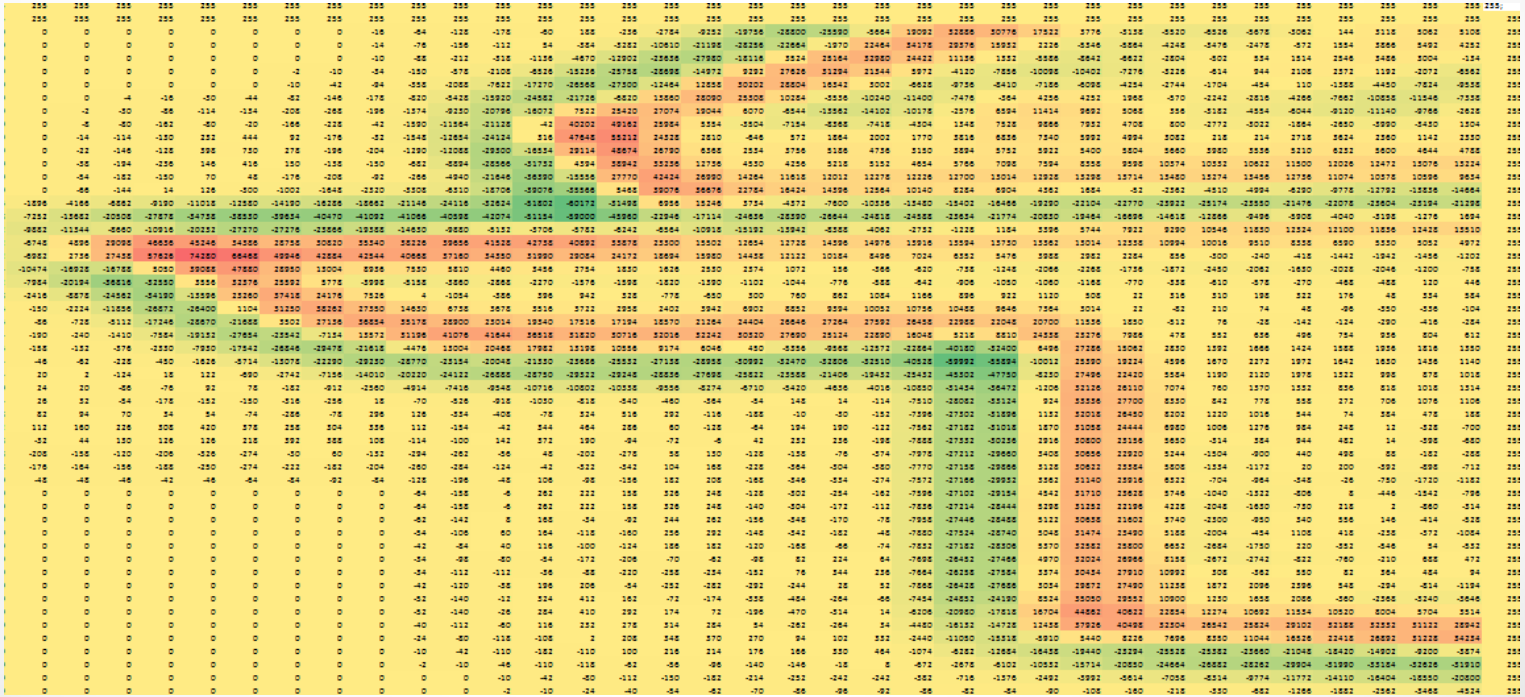
- Imaginea (Mat) care se obtine dupa ce se apeleaza computeLaplace contine valori pozitive si negative.
- Cele negative vor fi eliminate, iar cele pozitive contin elementele de interes.



-7252	-13682	-20508	-27878	-34738	-38530	-39634	-40470	-41092
-9882	-11344	-8660	-10916	-20232	-27270	-27276	-23866	-19388
-6748	4896	29098	46636	45246	34386	28758	30820	35340
-6982	2736	27438	57626	74280	66468	49946	42884	42544
10474	-16928	-16788	5050	39088	47880	28950	13004	8936
-7984	-20194	-36816	-32550	3556	32376	25592	5778	-3998
-2416	-8878	-24562	-34190	-13596	23260	37418	24176	7526
-150	-2224	-11856	-26872	-26400	1104	31250	38262	27350
-86	-728	-5112	-17246	-28670	-21688	3502	27136	36854
-190	-240	-1410	-7584	-19132	-27654	-23542	-7134	13572
-158	-132	-376	-2330	-7930	-17542	-26846	-29478	-21618

Desenare contururi

- Imagine de ansamblu asupra valorilor care se gaseau in coltul din acoperisul casei de mai devreme.



Filtrul Laplace

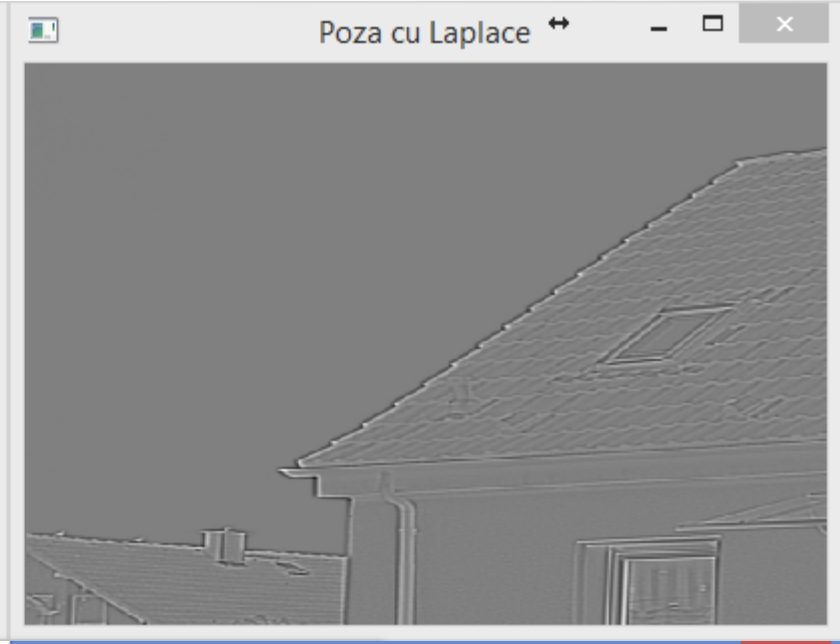
- Ultima metoda din clasa LaplacianZC
- Folosita pentru a detecta valorile negative

```
/* Calculam o imagine binara in care detectam pixelii
 * ce reprezinta margini - sunt langa pixeli negativi.
 * Daca produsul a doi pixeli adiacenti este mai mic
 * decat pragul, se marcheaza cu negru pixelul din
 * imaginea rezultat.
 */
Mat deseneazaContururi(float prag = 1.0)
{
    Mat_<float>::const_iterator it = laplace.begin<float>()+laplace.step1();
    Mat_<float>::const_iterator itend = laplace.end<float>();
    Mat_<float>::const_iterator itup = laplace.begin<float>();

    // Initializam imaginea binara de iesire cu alb
    Mat binary(laplace.size(), CV_8U, Scalar(255));
    Mat_<uchar>::iterator itout = binary.begin<uchar>()+binary.step1();

    // negam valoarea prad de intrare
    prag *= -1.0;
    for ( ; it!= itend; ++it, ++itup, ++itout)
    {
        // daca produsul a doi pixeli adiacenti este negativ
        // atunci avem o margine care trebuie desenata
        if (*it * *(it-1) < prag)
            *itout= 0; // marcare fiindca aveam pe orizontala valoare negativa
        else
            if (*it * *itup < prag)
                *itout= 0; // marcare fiindca aveam pe verticala valoare negativa
    }
    return binary;
}
```

main.cpp

- Am inclus aici si extragerea sectiunii din imagine si afisarea numerelor din ROI.
- getLaplacianImage si deseneazaConturur pot primi diferite valori pentru parametri.

```
void filtruLaplace(Mat &poza)
{
    /* Obtinem rezultatul Laplacian intr-o imagine pe 8 biti
     * 0 corespunde unui gri cu valoarea 128.
     * Daca nu este stabilita o scala de transformare,
     * valoarea maxima va fi 255.
     * Apelam intai computeLaplacian, apoi getLaplacianImage.
     */
    LaplacianZC laplacian;
    laplacian.setAperture(7);
    Mat flap = laplacian.computeLaplacian(poza);
    Mat pozaLaplace= laplacian.getLaplacianImage();

    //Extragem numerele de la o sectiune din poza
    //pentru a le putea vedea.
    Rect r = Rect(125,430,60,60);
    rectangle(flap, r, Scalar(255,0,0), 2, CV_AA);

    Mat pozaROI = flap(r);

    ofstream myfile;//le scriem intr-un fisier
    myfile.open ("E:\\Test\\numerepoza.txt");
    myfile << pozaROI;
    myfile.close();

    namedWindow("Poza cu Laplace flap", WINDOW_NORMAL);
    resizeWindow("Poza cu Laplace flap", 400, 280);
    imshow("Poza cu Laplace flap", flap);

    namedWindow("Poza cu Laplace", WINDOW_NORMAL);
    resizeWindow("Poza cu Laplace", 400, 280);
    imshow("Poza cu Laplace", pozaLaplace);

    Mat marginiLaplace = laplacian.deseneazaContururi(0.9f);
    namedWindow("Poza cu Laplace margini", WINDOW_NORMAL);
    resizeWindow("Poza cu Laplace margini", 400, 280);
    imshow("Poza cu Laplace margini", marginiLaplace);
}
```



- Filtrul detecteaza toate marginile, si cele importante si cele neimportante.
- Este sensibil la zgomot.

Filtrul bilateral

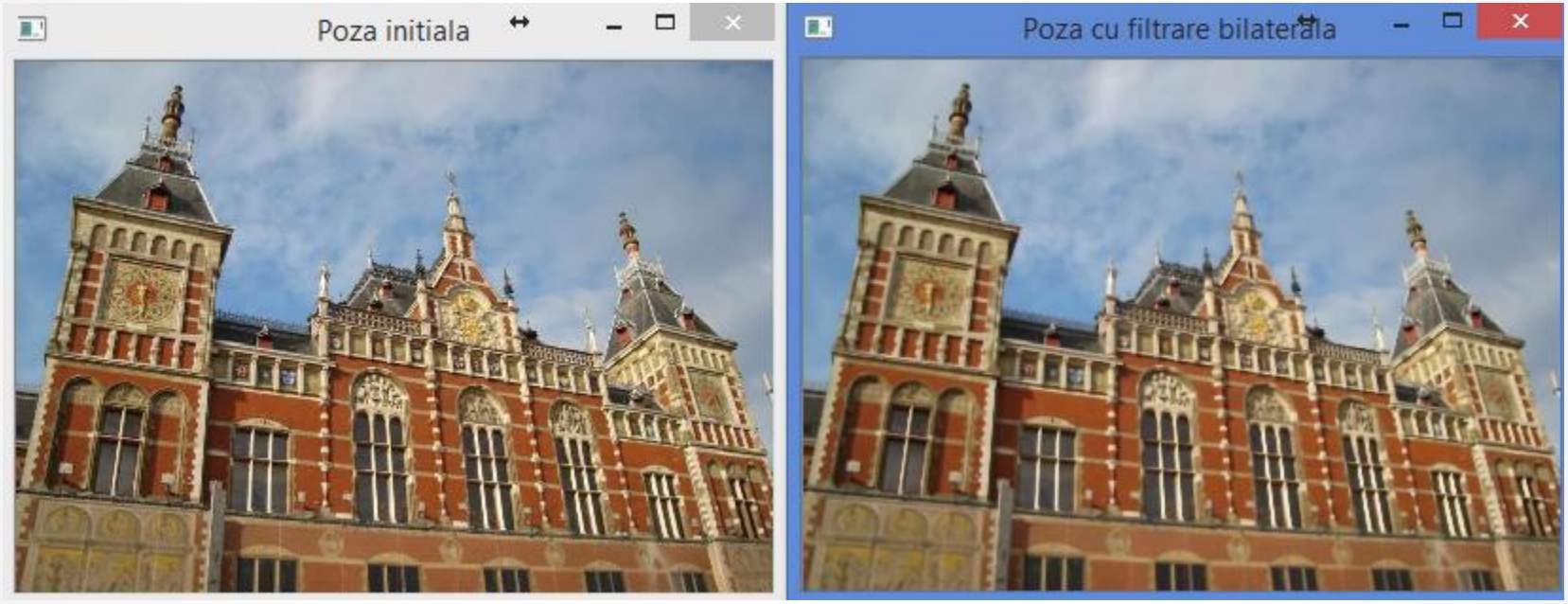
- Reduce zgomotul unei imagini, dar pastreaza marginile.
- Filtrarea nu depinde doar de distanta Euclidiana dintre pixeli, dar si de diferente intre intensitati de culori, de profunzime.
- Dureaza mai mult procesarea insa.

```
void bilateral(Mat &poza)
{
    Mat pozaBilaterala;

    bilateralFilter(poza, pozaBilaterala, 4, 200, 80);

    namedWindow("Poza initiala", WINDOW_NORMAL);
    resizeWindow("Poza initiala", 400, 280);
    imshow("Poza initiala", poza);

    namedWindow("Poza cu filtrare bilaterala", WINDOW_NORMAL);
    resizeWindow("Poza cu filtrare bilaterala", 400, 280);
    imshow("Poza cu filtrare bilaterala", pozaBilaterala);
}
```

Filtrul bilateral

- **void bilateralFilter(InputArray src, OutputArray dst, int d, double sigmaColor, double sigmaSpace, int borderType=BORDER_DEFAULT)**
  - **src si dst** – imaginile de intrare/iesire (pot fi color)
  - **d** - diametrul vecinatatii pentru fiecare pixel. Daca d < 0, se calculeaza din sigmaSpace
  - **sigmaColor** - sigma in spatiul culorilor. O valoare mai mare = culori mai indepartate din vecinatatea pixelului vor fi mixate
  - **sigmaSpace** - sigma in spatiul coordonatelor. O valoare mai mare = pixeli mai indepartati vor influenta daca au culori apropiate in functie de sigmaColor
  - **borderType** - afecteaza pixelii din margine.
    - optiuni; BORDER_DEFAULT, BORDER_REFLECT, BORDER_REPLICATE, BORDER_TRANSPARENT, BORDER_REFLECT_101

Filtrul bilateral



Proiecte 1/2

1. Faceti un proiect care sa citeasca o imagine si sa ii aplice acesteia o filtrare cu diferite valori pentru kernel. Sa se afiseze pe imagine ce marimi de kernel s-au utilizat.
   - Pentru a scrie pe imagine se utilizeaza putText.
2. Faceti un proiect cu GUI care sa permita sa se modifice toti parametrii unui anumit tip de filtrare pentru o imagine citita. Imaginea rezultata trebuie sa se poata salva.

Proiecte 2/2

3. O aplicatie cu GUI care sa foloseasca filtrul Sobel pentru a gasi marginile dintr-o imagine de intrare. Valorile pentru parametrii de intrare trebuie sa se poata stabili de catre utilizator.
4. Aplicati pentru frame-urile unui clip video un filtru Sobel pentru a obtine un clip in care fiecare frame contine numai marginile din clipul initial.