Corner detection

Narita Catalin-Ioan, *30235*

Data 31.05.2017

# Introducere

Intr-o imagine, colturile reprezinta o multime de informatii importante. Un colt poate fi definit ca intersectia a doua muchii. O alta definitie spune ca un colt este un punct in vecinatatea carua exista doua directii de muchii diferite si dominante.

In cadrul intersectiei a doua muchii, gradientul imaginii in amandoua directiile date de muchiile respective are o variatie mare, deci poate fi folosit pentru detectia acestor reguini.

Detectia colturilor este o abordare folosita in procesarea imaginilor. Este folosita frecvent in detectia miscarilor, modelare 3D, recunoasterea obiectelor etc.

Obiectivul aceste teme este de implementarea a trei algoritmi diferiti de detectie a colturilor.

# Studiul bibliographic

O metoda de detectie a colturilor este algoritmul lui Wang si Brandy. Acest algoritm considera o imagine ca fiind o suprafata si cauta locuri in imagine unde muchiile isi schimba brusc directia. O sugestie a celor doi este ca ar trebui folosita o metoda de smoothing pentru a reduce zgomotul din imagine(de exemplu Gaussian blur).

O alta metoda de detective a colturilor este algoritmul lui Moravec. Este unui dintre cei mai vechi algoritmi. Algoritmul testeaza fiecare pixel din imagine pentru a vedea daca este colt, luand in considerare cat de similara este o bucata din imagine cu centrul in piexlul respective cu alte bucati apropiate mai mar ice se suprapun. Aceasta similaritate este masurata luand in considerare suma patratelor diferentelor dintre pixelii celor doua bucati.

# Soluţia propusă

**Harris corner detection**

Primul algoritm folosit este algorimtul Harris de detectie a colturilor. Acesta urmareste zonele in care gradientul imaginii variaza foarte mult. Algoritmul calculeaza o matrice de referinta cu urmatoarea formula:

R = det(M) – k \* (trace(M))^2

Matricea M este o matrice a variatiei intensitatilor. Pasii care terbuie urmati pentru determinarea valorii aceste matrici sunt urmatorii:

1. Calculul derivatelor partiale in x si y pentru matricea initiala. Acest lucru se realizeaza folosind functia Sobel din OpenCV.
2. Pe baza derivatelor obtinute mai sus se calculeaza componentele ce alcatuiesc matricea M.

Aceste componente sunt: patratul fiecarei derivate si produsul derivatelor.

1. Din cauza faptului ca derivatele imaginilor produc zgomot, acesta tebuie eliminat. Pentru realizarea acestui lucru se foloseste functia GaussianBlur.

In momentul acesta am determinat matricea M. Aceasta are urmatoarea forma:



M =

Pasul urmator este cel de calculare a matricii de referinta R. Pentru realizarea acestui lucru

vom avea nevoie sa calculam determinantul si urma matricii M. Observam din matricea de mai sus urmatoarele:

det(M) = Ix^2 \* Iy^2 – (Ix \* Iy)^2

trace(M) = Ix^2 + Iy^2

In urma acestor operatii, matricea destinatie nu este in domeniul 0-255, deci va trebui sa o normalizam. Aceasta operatie poate fi realizata cu ajutorul functiei normalize().

Dupa aceste etape nu ne ramane decat sa parcurgem matricea initiala si sa comaram valoarea corespunzatoare din matricea calculata anterior cu un threshold. Daca aceasta valoare este mai mare decat valoarea de prag, inseamna ca am gasit un colt, pe care il marcam cu un cerc pe imaginea initiala cu ajutorul functiei circle().

**Shi-Tomasi corner detection**

Cel de-al doilea algoritm ales pentru a rezolva aceasta problema este algoritmul Shi-Tomasi. Este asemanator cu algoritmul Harris pana in punctul in care se determina matricea de referinta R. Aceasta nu mai este calculata cu formula R = det(M) – k \* (trace(M))^2, ci sunt folosite valorile proprii ale matricii M. In acest caz, matricea R se va calcula in felul urmator:

R = min(λ1, λ2).

Pentru acest lucru ne vom folosi de forma ecuatiei de gradul doi data de relatiile lui Viete:

X^2 – Sx + P = 0;

Valoarea S este defapt urma matricii M, iar P determinantul ei.

Pasul urmator este de calculare a discriminantului ecuatiei de gradul doi, iar apoi vor trebui calculate cele doua solutii, dupa care se alege maximul dintre ele.

Matricea de referinta contine maximul dintre cele doua solutii pentru fiecare punct din matricile de determinant si trace.

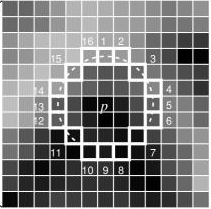
Pasii urmatori sun aceiasi ca si in cazul Harris, incepand cu operatia de normalizare.

**Fast corner detection**

Ultimul algoritm ales de mine este Fast corner detection. Algoritmul a fost propus de Edward Rosten si Tom Drummond.

Pasii algoritmului sunt urmatorii:

1. Se parcurge imaginea si se testeaza fiecare punct daca este un punct de interes sau nu. Pentru aceasta avem nevoie de intensitatea in acest punct.
2. Se alege o valoare de prag.
3. Consideram apoi un cerc de 16 pixeli cu pixelul din centru corespunzand cu punctul ales anterior.



1. Punctul p va fi considerat colt daca exista un set de n pixeli in cercul determinat care sunt toti mai deschisi decat Ip + t sau mai inchisi decat Ip – t, unde Ip reprezinta intensitatea punctului, iar t reprezinta pragul ales.

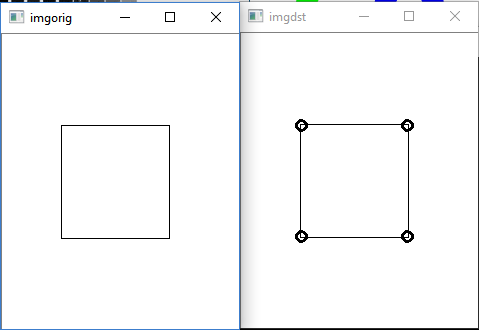
Pentru aceasta solutie a fost propusa o varianta in care se testeaza doar picelii 1,9,5 si 13(prima data 1 si 9; daca acestia sunt prea deschisi sau prea inchisi se testeaza 5 si 13). Punctul p este un colt daca cel putin 3 dintre aceste puncte sunt toate mai deschise decat Ip + t sua mai inchise decat Ip – t. Daca nu ne incadram in nici unul dintre aceste cazuri, atunci punctul nu poate fin un colt.

# Rezultate experimentale

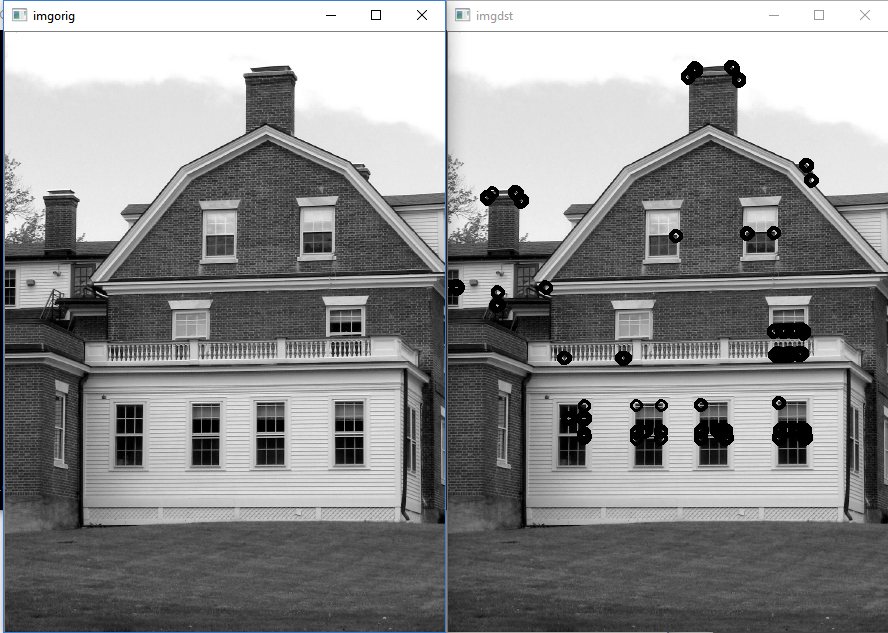
Algorimtii au fost testati atat pe imagini custom in care se gasesc doar colturi(ex: un patrat pe fundal alb), cat si pe imagini complexe.

**Harris:**

Threshold = 240

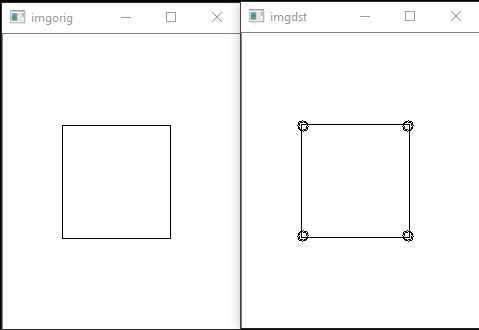


Threshold = 128

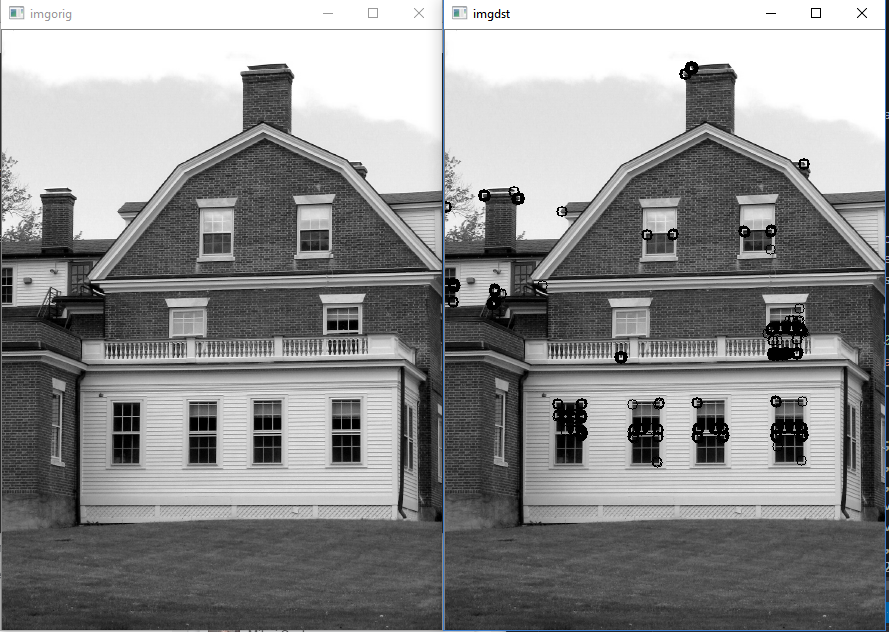


Shi-Tomasi:

Threshold = 240

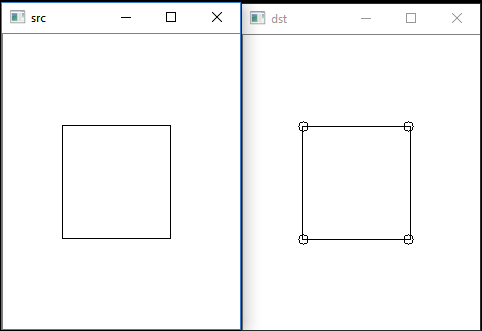


Threshold = 128

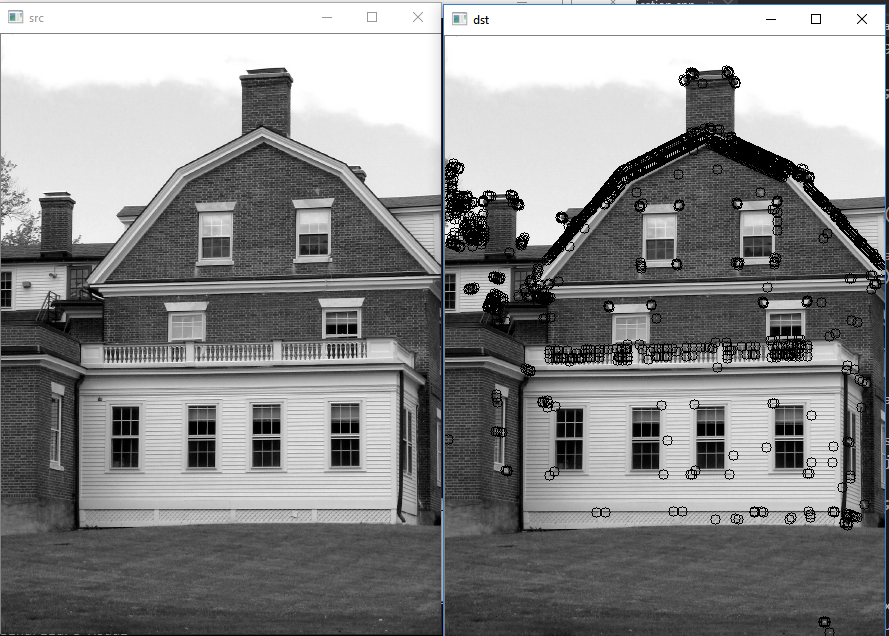


**Fast:**

Threshold = 50



Threshold = 50



# Concluzii

Valoarea parametrului de prag nu poate fi una fixa pentru diferite tipuri de imagini, deci acesta trebuie ales in functie de ce input avem. Chiar si asa, unele zone din imaginea complexa sunt marcate ca fiind colturi chiar daca nu ar trebui.

Acest lucru se poate observa cel mai bine totusi pe imaginea cu patratul, in care in fiecare colt sunt detectate colturi multiple.

Ca dezvoltari ulterioare as putea sa mentionez doua scenarii:

1. Algoritmul fast ar putea fi dezvoltat astfel incat sa testeze toate punctele de pe cerc(am pus in capitolul de anexe o functie care incearca sa faca acest lucru, dar nu functioneaza corespunzator).
2. Cea de-a doua modificare ar fi o metoda de selectie a valorii threshold-ului in functie de imagine.

# Bibliografie

<https://en.wikipedia.org/wiki/Corner_detection>

<http://docs.opencv.org/2.4/doc/tutorials/features2d/trackingmotion/harris_detector/harris_detector.html>

<http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_shi_tomasi/py_shi_tomasi.html>

<http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_fast/py_fast.html>

**Anexe**

**void fastCornerDetection() {**

**Mat src = imread("Images/123.png", CV\_LOAD\_IMAGE\_GRAYSCALE);**

**Mat dst = src.clone();**

**int threshold = 50;**

**int darker = 0;**

**int brighter = 0;**

**int cornerCount = 0;**

**Point2i point = src.at<uchar>(0,0);**

**int di[] = { -3,-3,-2,-1,0,1,2,3,3,3,2,1,0,-1,-2,-3 };**

**int dj[] = { 0,1,2,3,3,3,2,1,0,-1,-2,-3,-3,-3,-2,-1 };**

**for (int i = 0; i < src.rows; i++) {**

**for (int j = 0; j < src.cols; j++) {**

**darker = 0;**

**brighter = 0;**

**for (int k = 0;k<15;k++){**

**if (testPoint(point,i,j)) {**

**if (inBounds(src, di[k], dj[k])) {**

**if (src.at<uchar>(i, j) + threshold < src.at<uchar>(di[k], dj[k])) {**

**brighter++;**

**}**

**if (src.at<uchar>(i, j) - threshold > src.at<uchar>(di[k], dj[k])) {**

**darker++;**

**}**

**}**

**}**

**}**

**if ((brighter > 4 && darker == 0) || (darker > 4 && brighter == 0)) {**

**circle(dst, Point(j, i), 5, Scalar(0), 1, 8, 0);**

**cornerCount++;**

**point = Point2i(i, j);**

**}**

**}**

**}**

**imshow("src", src);**

**imshow("dst", dst);**

**cout << cornerCount;**

**waitKey(0);**

}