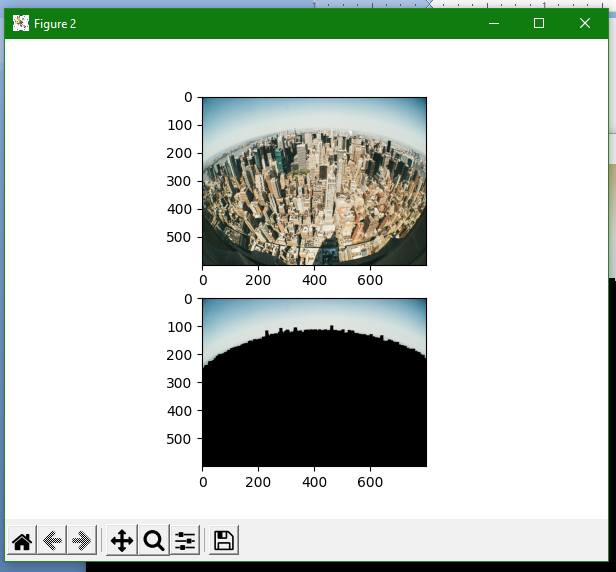
**Detectarea Liniei Orizontului in imagini de tip FishEye**

**Descrierea Problemei**

In problma abordata trebuie sa gasesc o metoda de a delimita cerul de orice altceva in imagine, iar ca si caz de testare am ales imagini de tip FishEye deoarece sunt mai complexe decat o imagine normala.

Pentru a reusi sa detectez cerul in aceasta problema am ales o abordare diferita, am ales sa colorez tot continutul unei imagin in afara de cer cu culoarea neagra aplicand o masca peste imaginea originala , astfel imaginea in urma procesarii arata astfel :



**Implementare :**

Crearea functiei principale numita **detect\_sky()** care primeste ca prametru imaginea citita. Ca prim pas in acest proces trebuie sa convertim imaginea in **grayscale**  pentru a realiza acest lucru am folosit functia **cvtColor(imagineaCitita,cv2.COLOR\_BGR2GRAY)** din openCV.

Pentru a reduce zgomotul din imagine avem doua optiuni cea simpla prin carea doar apelam functia **cv2.blur()** care ne aplica un filtru pe imaginea noastra, iar mai apoi pentru un rezultat mai precis trebuie sa aplicam si **cv2.medianBlur()**  prin care calculam media pe o anumita raza. Cea de-a doua abordare si cea aleasa de mine deoarece ofera mai multa versatilitate la partea de testatea este cea prin care ne cream un o functie numita **gaussianKernel()** prin care specificam dimenisunea si daca dorim sa il cream pe 2 dimnesiuni adica (X,Y).

Info

Un kernel mai poate fi denumit matrice de convolutie sau masca, reprezinta o matrice mica bidimensionala care contine valora pixelilori din jur pentru a calcula valoarea intensitatii pixelului actual. De obicei Kernel-ul este o matrice patratica de dimensiuni impare.

Convolutia este procedeul prin care din imaginea citita luam o submatrice de dimensiunea Kernel-ului pentru a inmulti fieacre element al submatricii cu elementul corespunzator din kernel, valorile rezultate le adunam si le normalizam astfel incat sa reprezinte o valoare de pixel.

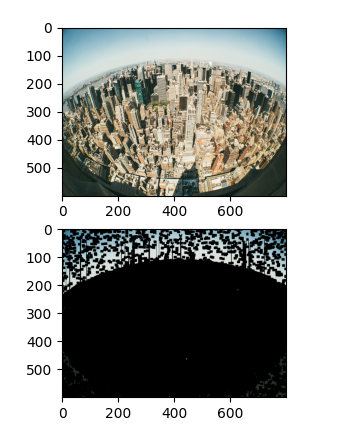
Rezultatul in urma apelului il salvam intr-o variabila numita **gaussin\_kernel\_x** iar din aceasta pentru a afla si cea de-a doua valoare pentru directia Y salvam in variabila **gaussin\_kernel\_y** transpusa folosind functia **.reashape(-1,1)** pe valoarea salvata in **gaussin\_kernel\_x** .

Pentru a aplica noul filtru creat de blur Gaussian folosim functia din openCv **cv2.filter2D(img\_gray,-1, gaussin\_kernel\_x)** cu ajutorul careia mai intai aplicam filtrul pentru directia **X,**  iar pentru **Y** apelam din nou functia anterioara doar ca de data aceasta putin diferit deoarece ca prim parametru luam imaginea rezultata anterior iar ca filtru punem transpusa calculata **cv2.filter2D(img\_gauss\_x,-1,gaussin\_kernel\_y)**

Functia **gaussianKernel()** a fost implementata sa suporte generarea de kernel bidimensional sau unidimensioanl in urma testelor am ales un kernel bidimensional deoarece dupa procesarea imaginii rezultatul este mult mai clar.

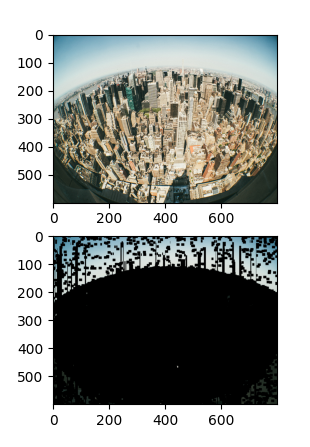
Pentru exemplificare voi atasa valorile data functiei pentru generarea de kernel dar si rezultatul procesarii:

**Kernel unidimensional :** gaussianKernel(3,1, False)



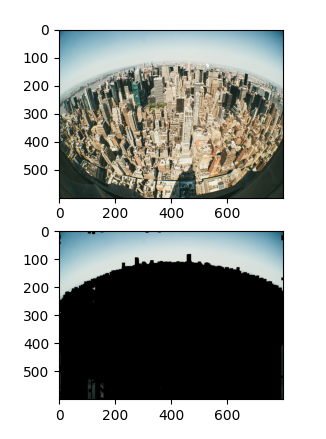
In urma acestui apel putem observa ca imaginea finala are un zgomot foarte mare partea cerului, acesta find neclar, iar linia orizontului nu foarte bine definite.

**Kernel unidimensional dupa ajustarea dimenisunii kernelului :** gaussianKernel(9,1, False)

****

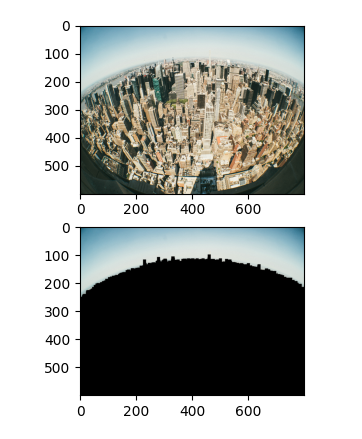
In urma acestei modificari putem observa imbunatatiri dar tot nu sunt multumitoare, iar preformanta progamului scade deoarece se foloseste pentru calcule o matrice de 9x9 in loc de una 3x3

**Kernel unidimensional dupa ajustarea valorii sigma:** gaussianKernel(9,3, False)



Dupa modificarea valorii din sigma putem observa imbunatatiri notabile, linia orizontului se poate distinge mult mai usor.

**Kernel bidimensional:** gaussianKernel(3,1, True)



Daca trecem la kernelul bidimensional cu valorile minime putem observa ca toate imperfectiunile din cer au disparut , linia orizontului se poate distinge, partea inferioara a imaginii este complet neagra, iar la capitolul performanta este o imbunatatire deoarce in kernel avem doar o matrice de 3x3.

Imaginea in urma aplicarii blur-ului Guassian



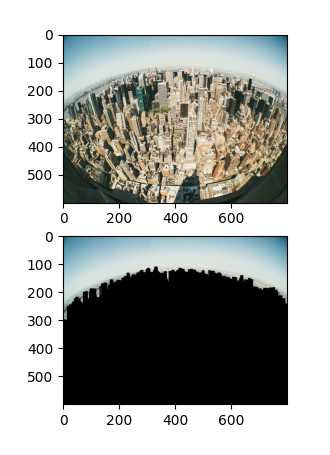
Ca pas urmator in rezolvarea problemei am decis sa aplic filtrul Laplacian din openCv cu ajutorul caruia sa detectez toate marginile din imagine pentru a determina la pasul urmator masca ce va trebui aplicata.

Functia apelata este **cv2.Laplacian(blured\_image,cv2.CV\_8U)** ne returneaza o imaginea ca cea de mai sus dar cu marginiile cladirilor evidentiate. Aceasta functie ia ca prim parametru imaginea create anterior iar cel de al doilea parametru ia **cv2.CV\_8U** aceast parametru spunandu-ne ca un pixel poate lua valroi intre [0-255] pentru acest parametru am testat si cu **cv2.CV\_32F** care poate lua valori intre 0-1.0 dar pentru a putea fi afisata corect imaginea trebuiesc inmultiti pixelii cu 255.

**Imaginea in urma aplicarii filtrului Laplacian**

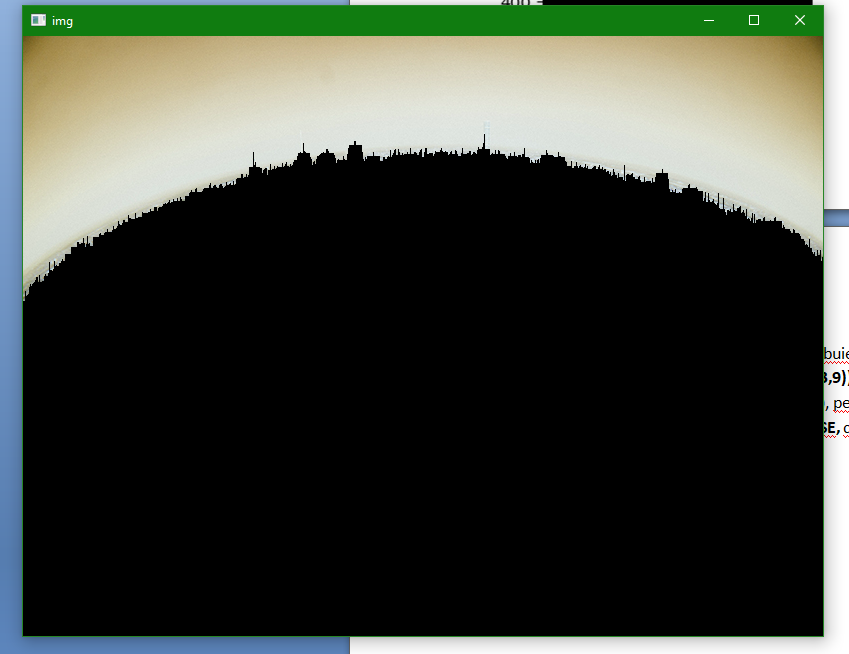


Ca prim pas in crearea mastii modificam tipul pixelilor mai mici ca 6 ii transformam in puncte de 8 biti. Valoarea a fost aleasa in urma mai multor teste deoarece daca folosim o valoare mult mai mica atunci va aparea zgomot in imagine la fel ca la filtrul de blur unidimensional, cea ce ne duce la un cer imperfect,dar daca luam o valoare mai mare de exemplu 15 puteam observa ca conturul orizontului nu mai este constant si masca intra in imagine, astfel in urma testelor valoarea optima pentru acest camp este 6.



Pentru a continua cu generarea masti mai intai trebuie sa ne generam un kernel nou cu ajutorul functiei **cv2.getStructuringElement(cv2.MORPH\_RECT,(13,9)) .** Acest kernel este folosit pentru acoperire cu ajutorul unor dreptunghiuri de dimensiunea (13,9), pentru aceasta functie in loc de **MORPH\_RECT** mai sunt disponible **MORPH\_CROSS** si **MORPH\_ELlIPSE,** dar in urma testelor am ales **MORPH\_RECT** deoarece are cea mai mare acoperire.

**MORPH\_CROSS**

****

**MORPH\_ELLIPSE**

****

**MORPH\_RECT**

****

In urma acestor teste am pastrat variant cu MORP\_RECT deoarce are o zona de acoperire mai mare ca celelate doua.

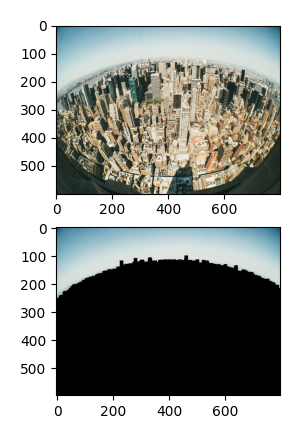
Ca pas urmator alteram masca creata prin functia **cv2.morphologyEx(gradient\_mask,cv2.MORPH\_ERODE,kernel)** in care primul parametru este masca create anterior, **MORPH\_ERODE** este modul in care va fi alterat filtrul iar **kernel-ul** este cel generat anterior.

Ca ultim pas cream o functie unde aplicam un filtru median pentru a normaliza valorile si salvam in masca dupa aplicarea filtrului toate valorile unse exista valoare 0 sau 1. Dupa ce noua masca este returnanta folosim functia **cv2.bitwise\_and(img, img, mask=mask)** care ia ca primi 2 parametri imaginea originala iar ca ultimo parametru masca creata aceasta functie are rolul de a suprapune masca create pe imaginea originala.

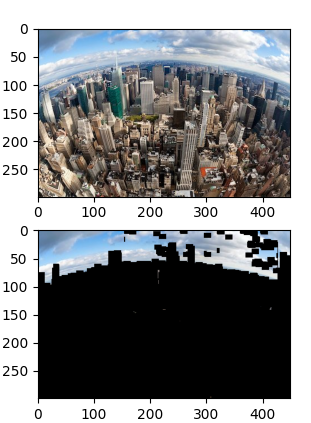
**Cazuri de test:**

Pentru test am ales un set de 8 poze pentru a observa performantele aplicatiei.

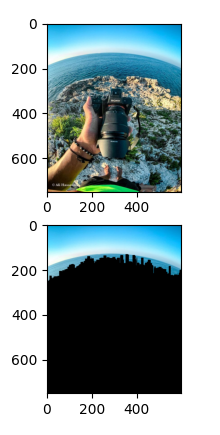
**Test 1:** O imagine cu cerul senin



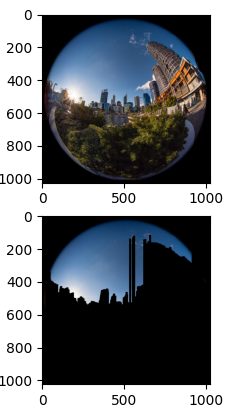
**Test 2:** O imagine cu cerul innorat – in acest caz apar cateva imperfectiuni in cer

****

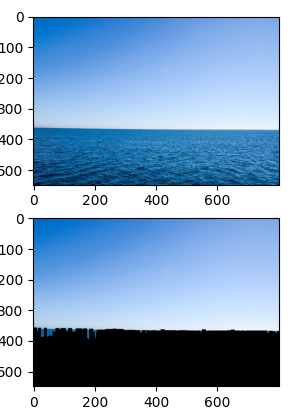
**Test 3:** O imagine cu linia orizontului pe mare – in cazul acesta linia orizontului este putin mai jos din cauza culorilor similar.



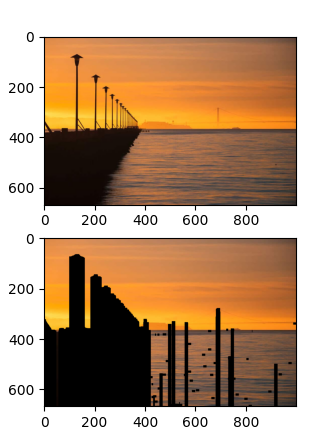
**Test 4:** O imagine rotunjita – programul detecteaza cerul correct,exclude toate cladirile, dar din cauza culorii negre a masti nu se mai observa jumatatea inferioara a cercului

****

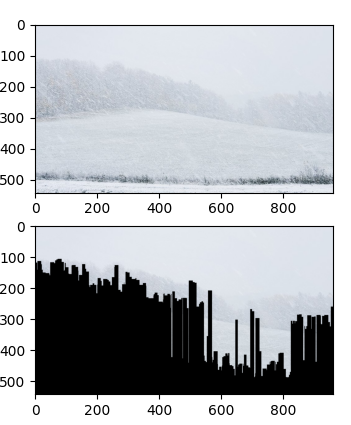
**Test 5:** O imagine normal cu linia orizontului – detectita linii este aproape perfecta desi programul nu a fost facut pentru astfel de linii

****

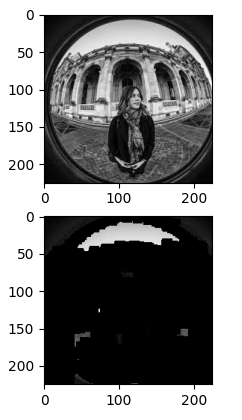
**Test 6:** O imagine in care cerul si marea nu se disting foarte bine – detectita linii este aproape inexistenta , programul reusind doar sa scoata obiectele din cadru.

****

**Test 7:** O imagine in cu zapada – putem observa ca detectia functioneaza doar partial din cauza lipsei de contrast

****

**Test 8:** O imagine grayscale – putem observa ca detectia functioneaza pe o imagine deja grayscale in care cerul este mai putin de 10% din imagine



**Concluzie:**

Pe baza testelor anterioare detectia are rezultate satisfacatoare in 50% din cazuri deoarece are nevoie pentru acuratete maxima ca cerul sa fie bine definit si senin, in urma testelor se poate observa ca daca cerul este senin si bine definit detectia functioneaza si pe imaginile normale nu numai pe FishEye.

**Webgrafie:**

[**https://www.askpython.com/python-modules/opencv-filter2d**](https://www.askpython.com/python-modules/opencv-filter2d)

[**https://docs.opencv.org/3.4/**](https://docs.opencv.org/3.4/)

**https://www.youtube.com/c/ProgrammingKnowledge**