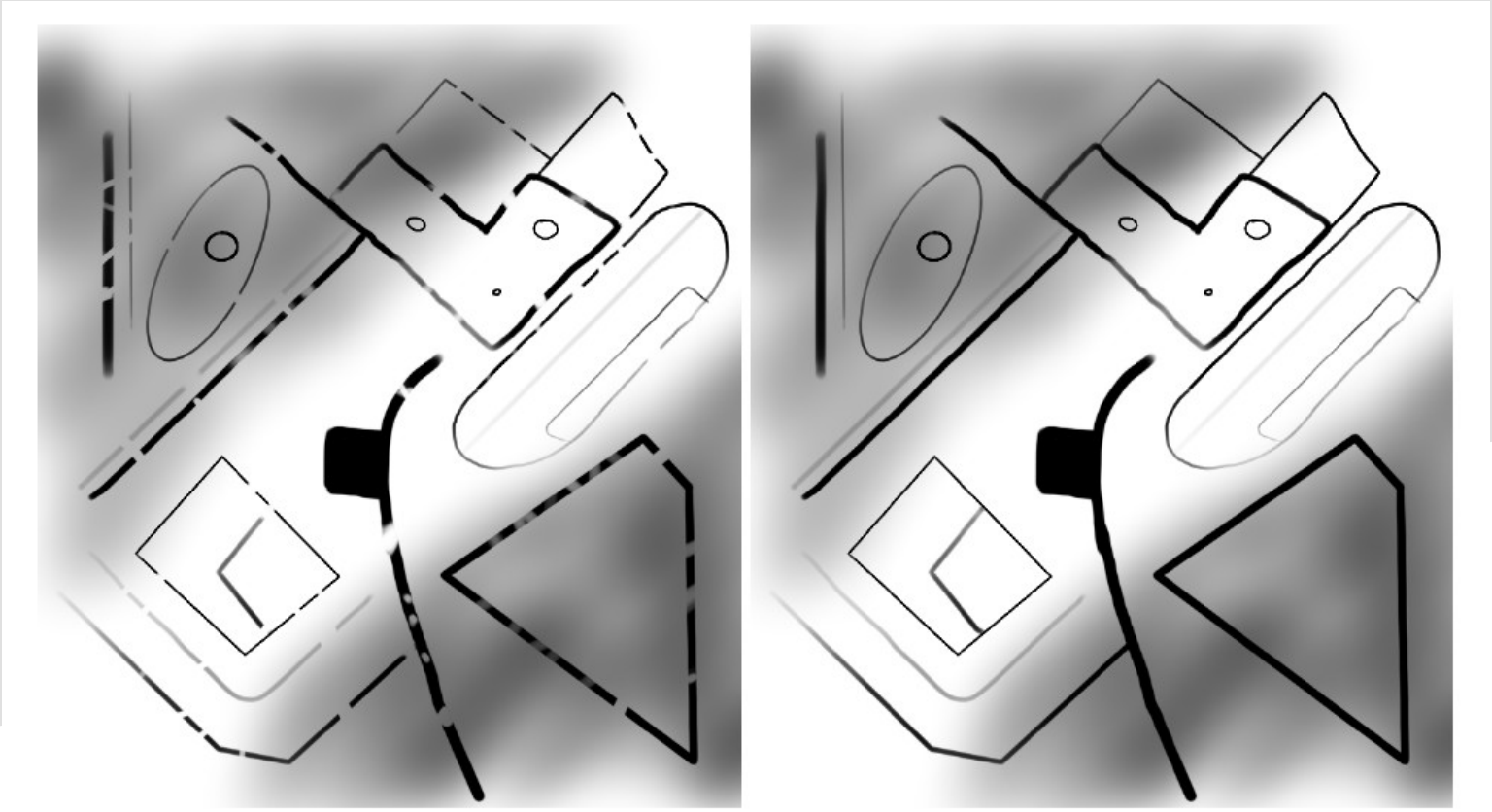
Реконструкция пунктиров

Студентка ФРТК
Дуденко Екатерина
Б01-001

Цель:



- Подсказка к решению:

  управляемый морфологический фильтр

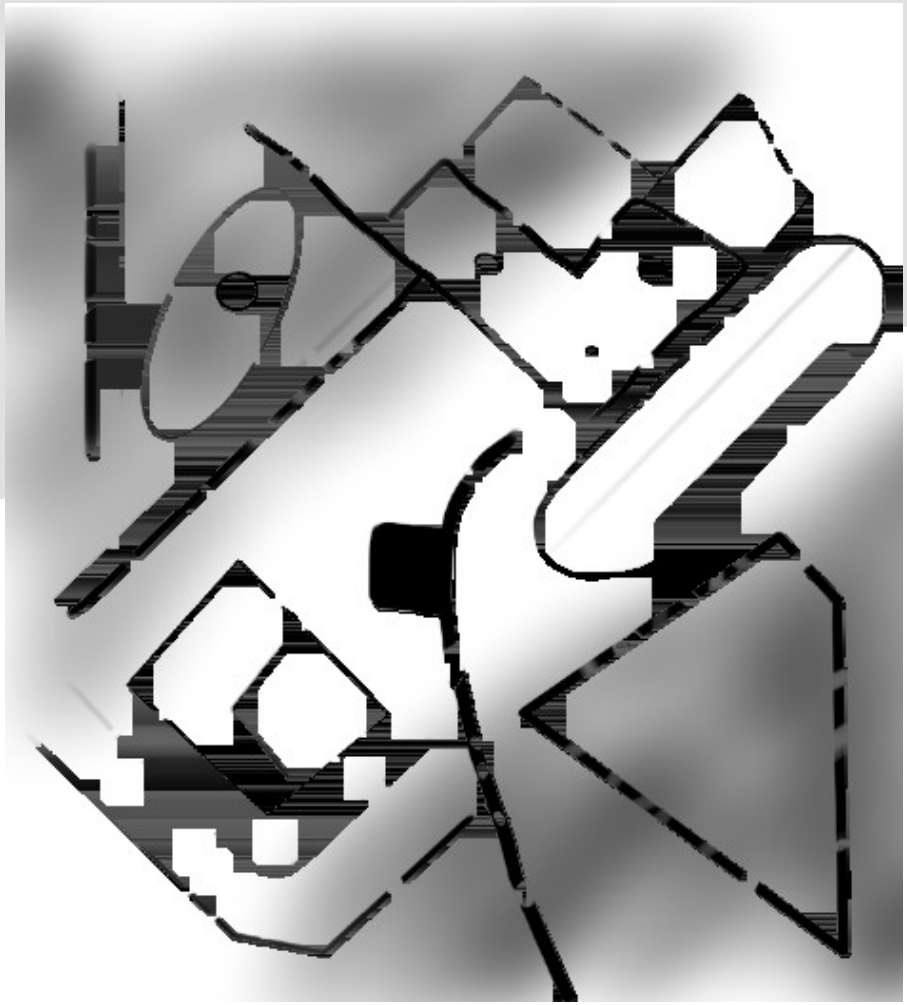
Морфологические фильтры

Erode — минимальное значение под ядром
Dilate — максимальное значение под ядром
Open = dilate(erode(src, kernel)
Close = erode(dilate(src, kernel)
Blackhat = close(src, kernel) - src

erode dilate open close blackhat

Первая попытка решения

```
def close_op(img, filterSize):
  kernel =
cv2.getStructuringElement(cv2.MORPH_RECT,
filterSize)
  ret = cv2.Canny(img, 100, 200)
  closing = cv2.morphologyEx(ret, cv2.MORPH_CLOSE,
     kernel)
  return closing


def main():
  img = cv2.imread('image.png')
  img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  closing = close_op(img, (20, 22))

  px = 50

  for i in range(img.shape[0]):
    for j in range(img.shape[1]):
      if closing.item(i, j) > 200:
        if img.item(i, j) < 100:
          px = img.item(i, j)
        else:
          img.itemset((i, j), px)
```



Поиск ориентации линий

```
def calcTenz(inputIMG, w):
    img = inputIMG.astype(np.float32) #to float

    imgDiffX = cv.Sobel(img, cv.CV_32F, 1, 0, 3)
    imgDiffY = cv.Sobel(img, cv.CV_32F, 0, 1, 3)
    imgDiffXY = cv.multiply(imgDiffX, imgDiffY)

    imgDiffXX = cv.multiply(imgDiffX, imgDiffX)
    imgDiffYY = cv.multiply(imgDiffY, imgDiffY)
    J11 = cv.boxFilter(imgDiffXX, cv.CV_32F, (w,w))
    J22 = cv.boxFilter(imgDiffYY, cv.CV_32F, (w,w))
    J12 = cv.boxFilter(imgDiffXY, cv.CV_32F, (w,w))

    tmp1 = J11 + J22
    tmp2 = J11 - J22
    tmp2 = cv.multiply(tmp2, tmp2)
    tmp3 = cv.multiply(J12, J12)
    tmp4 = np.sqrt(tmp2 + 4.0 * tmp3)
    lambda1 = 0.5*(tmp1 + tmp4)       # biggest eigenvalue
    lambda2 = 0.5*(tmp1 - tmp4)       # smallest
eigenvalue

    imgCoherencyOut = cv.divide(lambda1 - lambda2, lambda1
+ lambda2)
    imgOrientationOut = cv.phase(J22 - J11, 2.0 * J12,
angleInDegrees = The)
    imgOrientationOut = 0.5 * imgOrientationOut

    return imgCoherencyOut, imgOrientationOut
```

J = (J11 J12; J12 J22) - Tenzor

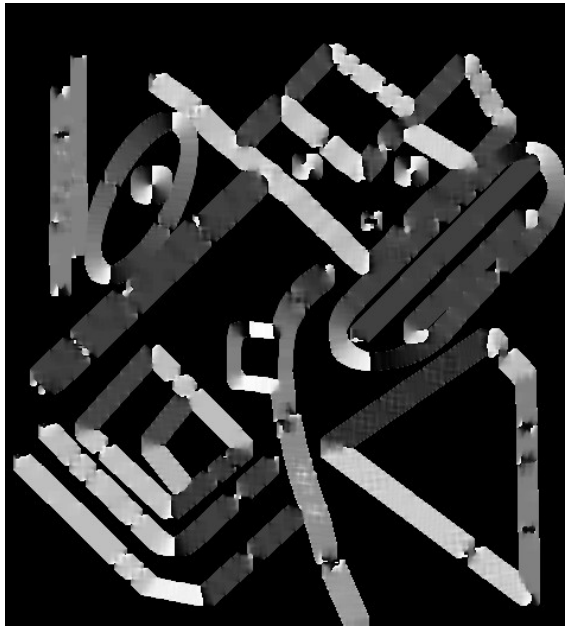
I1 = 0.5*(J11 + J22 + sqrt((J11-J22)^2 + 4*J12^2))

I2 = 0.5*(J11 + J22 - sqrt((J11-J22)^2 + 4*J12^2))

Собственные числа

tan(2*Alpha) = 2*J12/(J22 - J11)
Alpha = 0.5 atan2(2*J12/(J22 — J11))

Подсчет угла

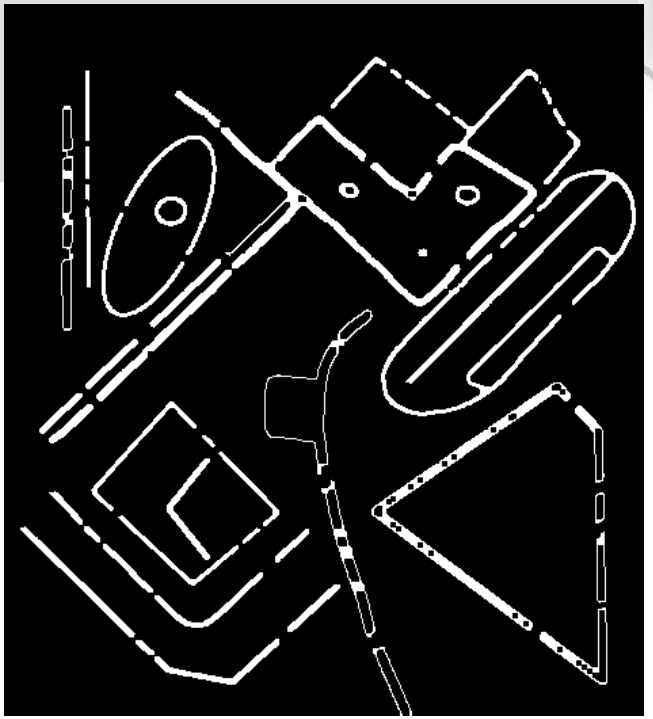


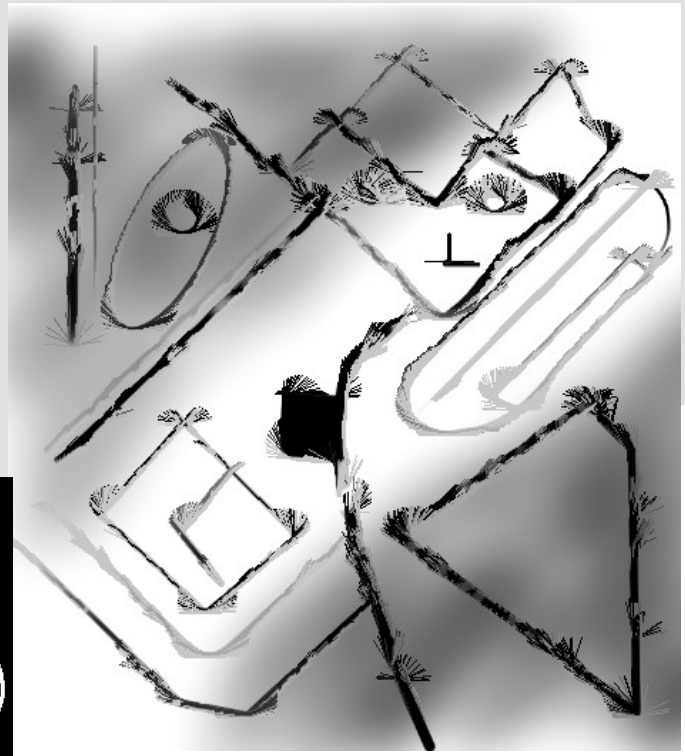
Вторая попытка решения

```
def paint_lines(filtIMG, leng,
orientationIMG, inIMG):


  for i in range(filtIMG.shape[0]):
    for j in range(filtIMG.shape[1]):
      if(filtIMG.item(i, j) > 0):


        start_point = (j, i)
        cos =
math.cos(orientationIMG.item(i, j) *
3.1415 + 3.1415/2)
        sin =
math.sin(orientationIMG.item(i, j) *
3.1415 + 3.1415/2)
        end_point = (int(line * sin) +
j, int(line * cos) + i)


        color = inIMG.item(i, j)
        if color > 200 :
          color = 200


       thickness = 1
        if j + line < filtIMG.shape[1]
and i + line <      filtIMG.shape[0] :
          inIMG = cv.line(inIMG,
start_point, end_point, color,
thickness)


  return inIMG
```

Первичная обработка для удаления серых пятен и дальнейшего определения направления линий.

canny(50,100) + close(4,4)





Результат, полученный путем рисования линий по известным направлениям.

Третья попытка решения

```
def get_kernel(angle, kern, ancx, ancy):
  cos = math.cos(angle * 3.1415 + 3.1415/2)
  sin = math.sin(angle * 3.1415 + 3.1415/2)
  kern[ancx][ancy] = 1;

  for i in range(1, kern.shape[0]):
    x = round(i * cos)
    y = round(i * sin)

    if x + ancx >= kern.shape[0] or y + ancy >=
kern.shape[1] or ancx + x < 0 or ancy + y < 0:
      return kern
    kern[x + ancx][y + ancy] = 1
    kern[ancx - x][ancy - y] = 1
  return kern

def paint_hole(kern, imgIn, i, j, color):
  for k in range(0, kern.shape[0]):
    for p in range(0, kern.shape[1]):
      im = i + k - ancx
      jm = j + p — ancy

      if im >= 0 and jm >= 0 and im < imgIn.shape[0]
and jm <    imgIn.shape[1]:
        if kern[k][p] != 0 :
          imgIn[im][jm] = 0
  return imgIn
```