图像处理第三次作业

王兆盟 3017218072

2019 年 12 月 9 日

1 问题一：对LoG(高斯——拉普拉斯算子)的数学形式进行数学推导

拉普拉斯算子是一种高通滤波器，是影像灰度函数在两个垂直方向二阶偏导数之和。在离散数字影像的情况下,直接用影像灰度级的二阶差分代替连续情形下的二阶偏导数,对噪声很敏感,在提取边缘时往往会出现伪边缘响应。为克服拉普拉斯算子的不足,宜先对数字影像进行低通滤波，抑制噪声。高斯函数是一种很好的归一化低通滤波器,可用于对数字影像进行低通滤波以减少噪声的影响，在此基础上再利用拉普拉斯算子提取边缘,这就是高斯-拉普拉斯算子，又称为LOG(Laplacian of Gaussian)算子。

1980年，Marr和Hildreth提出将Laplace算子与高斯低通滤波相结合，提出了LOG（Laplace and Guassian）算子。

1) 对图像先进性高斯滤波($G_\sigma \times f$)，再进行Laplace算子运算$\Delta(G_\sigma \times f)$

2) 保留一阶导数峰值的位置，从中寻找Laplace过零点

3) 对过零点的精确位置进行插值估计

Laplace算子作为一种优秀的边缘检测算子，在边缘检测中得到了广泛的应用。该方法通过对图像求图像的二阶倒数的零交叉点来实现边缘的检测，公式表示如下：

$$\nabla^2 f = \frac{\partial f}{\partial x^2} + \frac{\partial f}{\partial y^2}$$

由于Laplace算子是通过对图像进行微分操作实现边缘检测的，所以对离散点和噪声比较敏感。于是，首先对图像进行高斯卷积滤波进行降噪处

理，再采用Laplace算子进行边缘检测，就可以提高算子对噪声和离散点的鲁棒性，如此，拉普拉斯高斯算子Log（Laplace of Gaussian）就诞生了。

高斯卷积（Gaussian convolution ）,高斯函数的表达式如下：

$$G_\sigma(x,y)=\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{x^2+y^2}{\sigma^2}}$$

原图像与高斯卷积的表达式如下：

$$\Delta[G_\sigma(x,y)\times f(x,y)]=[\Delta G_\sigma(x,y)]\times f(x,y)=LoG\times f(x,y)$$

因为：

$$\frac{\mathrm{d}[h(t)\times f(t)]}{\mathrm{d}t}=\frac{\int f(T)h(t-T)\mathrm{d}T}{\mathrm{d}t}=f(T)\int\frac{\mathrm{d}}{\mathrm{d}t}h(t-T)\mathrm{d}T=f(t)\times\frac{\mathrm{d}h(t)}{\mathrm{d}t}$$

所以Log可以通过先对高斯函数进行偏导操作，然后进行卷积求解，公式表示如下：

首先考虑高斯函数的一阶偏导数：

$$\frac{\partial}{\partial x}G_\sigma(x,y)=\frac{\partial}{\partial x}e^{-\frac{x^2+y^2}{2\sigma^2}}=-\frac{x}{\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$$

然后计算二阶偏导数,如果忽略正则化系数（normalizing coefficient）$\frac{1}{\sqrt{2\pi\sigma^2}}$,我们可以得到化简形式：

$$\frac{\partial^2}{\partial^2 x}G_\sigma(x,y)=\frac{x^2}{\sigma^4}e^{-\frac{x^2+y^2}{2\sigma^2}}-\frac{1}{\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}=\frac{x^2-\sigma^2}{\sigma^4}e^{-\frac{x^2+y^2}{2\sigma^2}}$$

相似的，我们计算对y的偏导数

$$\frac{\partial^2}{\partial^2 y}G_\sigma(x,y)=\frac{y^2-\sigma^2}{\sigma^4}e^{-\frac{x^2+y^2}{2\sigma^2}}$$

最终我们得到LoG(高斯——拉普拉斯算子)或卷积核的定义如下：

$$LoG\triangleq\Delta G_\sigma(x,y)=\frac{\partial^2}{\partial^2 x}G_\sigma(x,y)+\frac{\partial^2}{\partial^2 y}G_\sigma(x,y)=\frac{x^2+y^2-\sigma^2}{\sigma^4}e^{-\frac{x^2+y^2}{2\sigma^2}}$$

2 实现最小二乘法、RANSAC法、霍夫变换法

2.1 对于直线方程y=ax+b，生成一系列纵坐标符合高斯分布的点，再人工加入一系列的outlier，使用上述三种方法拟合一条直线

2.1.1 最小二乘法

首先笔者编写了一个脚本，绘制了一条满足方程$y=5x+3$的由蓝色散

点构成的直线（除去添加的高斯噪声点和手动添加的outlier点之外），同时手动再向其中加入了纵坐标符合高斯分布的一些散点，以及10个在整个图片范围内随机位置出现的散点作为outlier点。代码如下：

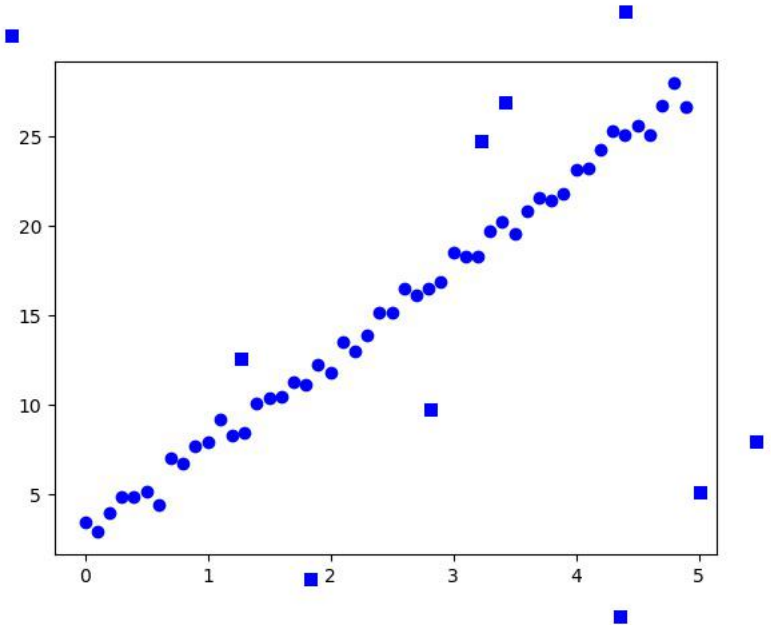
```
X = np.arange(0, 5, 0.1)
Z = [3 + 5 * x for x in X]
Y = [np.random.normal(z, 0.5) for z in Z]

plt.plot(X, Y, 'bo')
plt.savefig('./test.jpg')
img = Image.open("./test.jpg","r")
line_pixTuple_b = (0, 0, 255, 1)

for n in range(10):
    i = random.randint(0,img.size[0]-10)
    y = random.randint(0,img.size[1]-10)
    for j in range(10):
        for k in range(10):
            img.putpixel((i + j, y + k), line_pixTuple_b)

img.save("./test.jpg")
```

生成图片结果如下：



最小二乘法是一种数学优化技术。它通过最小化误差的平方和寻找数据的最佳函数匹配。利用最小二乘法可以简便地求得未知的数据，并使得这些求得的数据与实际数据之间误差的平方和为最小。我们在这里首先是在得到边缘点后，再使用最小二乘法对离散的散点进行直线的拟合。

笔者在此考虑到接下来在实际的简单真实图像（而非构造的直线图像）中的应用，因此在这里也提前做了图像的预处理，即先将图片进行灰度化，二值化再使用一阶偏导数的sobel算子对其进行卷积操作，得到了原图像的边缘点构成的图像，然后再使用最小二乘法拟合出图像中的直线。

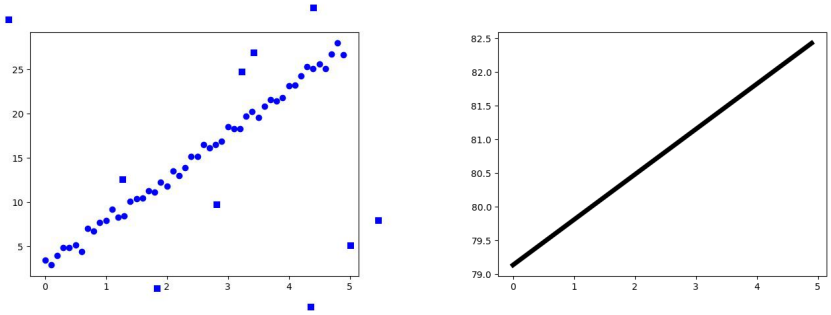
代码如下：

```
def least_square_method(img):
    frame = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    frame = cv2.GaussianBlur(frame, (3,3), 0)
    _, Imask = cv2.threshold(frame, 80, 255, cv2.THRESH_BINARY)
    Imask = cv2.erode(Imask, None, iterations=2)
    Imask = cv2.dilate(Imask, np.ones((5, 5), np.uint8), iterations=2)
    sobely=cv2.Sobel(Imask,cv2.CV_64F,0,1,ksize=5)
    sobely=abs(sobely)
    _,c=cv2.threshold(sobely, 500, 255, cv2.THRESH_BINARY)
    row,col=c.shape
    c=c[int(row/5):int(row*4/5),int(col/3):int(2*col/3)]
    row,col=c.shape
    cv2.imshow('o',c)
    x=np.linspace(0,col,col)
    y=np.array(x)
    for i in range(col):
        cy=row-np.argmax(c[:,i])
        y[i]=cy
    xmean=x.mean()
    ymean=y.mean()
    k=((x*y).mean()-xmean*ymean)/(pow(x,2).mean()-(xmean**2))
    b=ymean-k*xmean
    X = np.arange(0, 5, 0.1)
    Y = [b + k * x for x in X]
    plt.plot(X, Y, 'k',linewidth=5)
    plt.show()
```

图片效果如下：

2.1.2 RANSAC算法

RANSAC是“RANdom SAmple Consensus（随机抽样一致）”的缩写。它可以从一组包含“局外点”的观测数据集中，通过迭代方式估计数学模



(a) 原始图像　　(b) 最小二乘法得到直线

图 1: 最小二乘法检测直线

型的参数。它是一种不确定的算法——它有一定的概率得出一个合理的结果，为了提高概率必须提高迭代次数。

RANSAC算法的基本假设是样本中包含正确数据(inliers，可以被模型描述的数据)，也包含异常数据(outliers，偏离正常范围很远、无法适应数学模型的数据)，即数据集中含有噪声。这些异常数据可能是由于错误的测量、错误的假设、错误的计算等产生的。同时RANSAC也假设，给定一组正确的数据，存在可以计算出符合这些数据的模型参数的方法。

在我们的数据中包含局内点和局外点（outlier），其中局内点近似的被直线所通过，而局外点远离于直线。简单的最小二乘法不能找到适应于局内点的直线，原因是最小二乘法尽量去适应包括局外点在内的所有点。相反，RANSAC能得出一个仅仅用局内点计算出模型，并且概率还足够高。

笔者编写脚本，使用RANSAC算法，输入为:

data - 样本点

model - 假设模型:事先自己确定

n - 生成模型所需的最少样本点

k - 最大迭代次数

t - 阈值:作为判断点满足模型的条件

d - 拟合较好时,需要的样本点最少的个数,当做阈值看待

输出: bestfit - 最优拟合解（返回nil,如果未找到）

核心代码如下：

```
def ransac(data, model, n, k, t, d, debug=False, return_all=False):
    iterations = 0
```

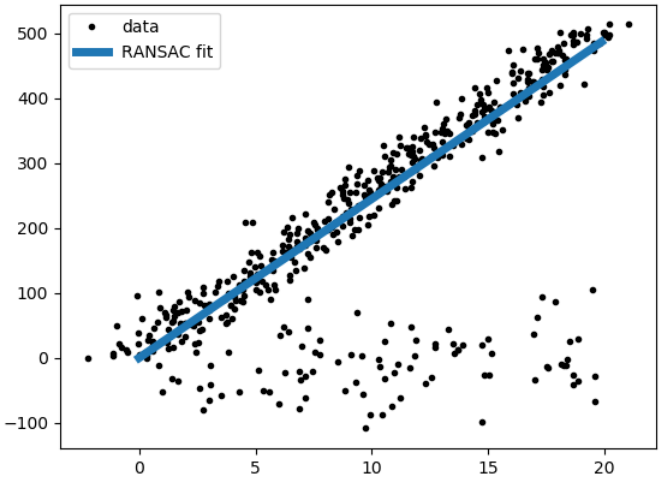
```
    bestfit = None
    besterr = np.inf
    best_inlier_idxs = None
    while iterations < k:
        maybe_idxs, test_idxs = random_partition(n, data.shape[0])
        maybe_inliers = data[maybe_idxs, :]
        test_points = data[test_idxs]
        maybemodel = model.fit(maybe_inliers)
        test_err = model.get_error(test_points, maybemodel)
        also_idxs = test_idxs[test_err < t]
        also_inliers = data[also_idxs, :]
        if debug:
            print('test_err.min()', test_err.min())
            print('test_err.max()', test_err.max())
            print('numpy.mean(test_err)', np.mean(test_err))
            print('iteration \%d:len(alsoinliers) = \%d' \% (iterations
                                                , len(also_inliers)))
        if len(also_inliers > d):
            betterdata = np.concatenate((maybe_inliers, also_inliers))
            bettermodel = model.fit(betterdata)
            better_errs = model.get_error(betterdata, bettermodel)
            thiserr = np.mean(better_errs)
            if thiserr < besterr:
                bestfit = bettermodel
                besterr = thiserr
                best_inlier_idxs = np.concatenate((maybe_idxs,
                                                   also_idxs))

        iterations += 1
    if bestfit is None:
        raise ValueError("did't meet fit acceptance criteria")
    if return_all:
        return bestfit, {'inliers': best_inlier_idxs}
    else:
        return bestfit
```

效果如下：

2.1.3 霍夫（Hough）变换法

霍夫变换是图像变换中的经典手段之一，主要用来从图像中分离出具有某种相同特征的几何形状（如，直线，圆等）。霍夫变换寻找直线与圆的方法相比与其它方法可以更好的减少噪声干扰。经典的霍夫变换常用来检测直线，圆，椭圆等。



直线检测为例，每个像素坐标点经过变换都变成都直线特质有贡献的统一度量，一个简单的例子如下：一条直线在图像中是一系列离散点的集合，通过一个直线的离散极坐标公式，可以表达出直线的离散点几何等式如下：$xcos(\theta) + ysin(\theta) = r$ 其中角度θ指r与X轴之间的夹角，r为到直线几何垂直距离。任何在直线上点，x, y都可以表达，其中 r， θ是常量。

笔者在此同样考虑到接下来在实际的简单真实图像（而非构造的直线图像）中的应用，因此在这里也提前做了图像的预处理，即先将图片进行灰度化，二值化再使用一阶偏导数的canny算子对其进行卷积操作，得到了原图像的边缘点构成的图像，然后再使用最小二乘法拟合出图像中的直线。

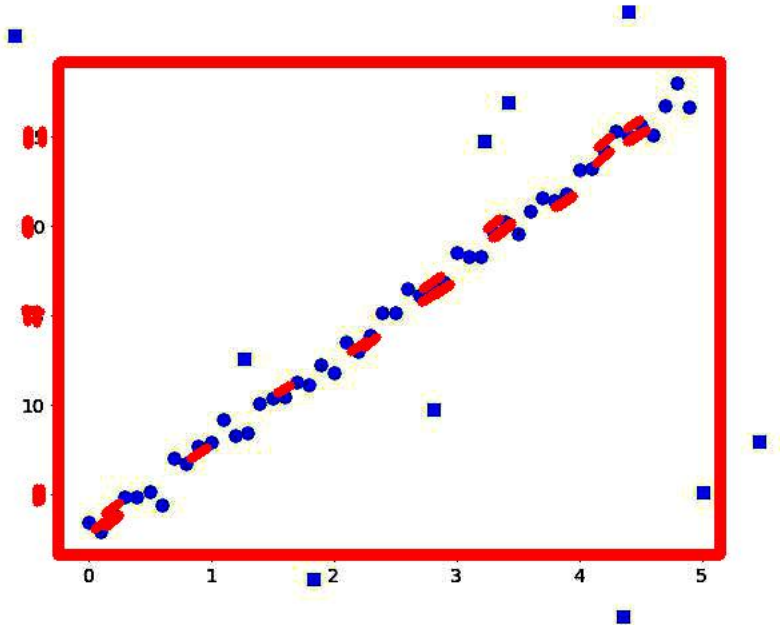
核心代码如下：

```
def hough_transform(img):
    img = ImageEnhance.Contrast(img).enhance(3)
    img = np.array(img)
    result = img_processing(img)
    lines = cv2.HoughLinesP(result, 1, 1 * np.pi / 180, 10,
                                        minLineLength=10, maxLineGap=5)
    print("Line Num : ", len(lines))

    for line in lines:
        for x1, y1, x2, y2 in line:
```

```
            cv2.line(img, (x1, y1), (x2, y2), (255, 0, 0), 5)
        pass
    img = Image.fromarray(img, 'RGB')
    img.save("hough_detect.jpg")
```

效果如下(其中红色为检测到的直线)：



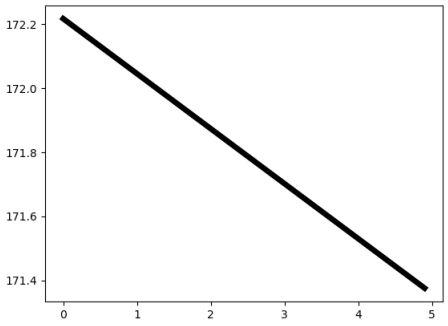
3 找到一幅实际图像（较简单的），使用一阶导数或二阶导数找出边缘点，使用上述三种方法，找到其中的直线

由于笔者已经在之前编写了对于实际图像的预处理，将图片进行灰度化，二值化再使用一阶偏导数的算子对其进行卷积操作，得到了原图像的边缘点构成的图像，接着只需直接输入实际图像进行测试即可。

笔者选取了一张较为笔直有较多直线的公路照片作为实际图像：

测试结果如下：

1. 最小二乘法



2. 霍夫变换法

对比上述结果可知，采用最小二乘法对图片中在反对角线附近，原图片中左侧的山坡和右侧公路的边界组成的一条近似直线进行了较为准确的拟合。而采用霍夫变换法则对原图片中多达567条可能的直线进行了检测，当然这其中也包含了较多误判的地方。