Programming Assignment 1
图像滤波与Hough变换

在这项任务中，你将实现一些基本的图像处理算法，并把它们放在一起，建立一个基于Hough变换的线条检测器。你的代码将能够把线变成图像中的边缘。我们已经为你提供了一些图像来测试你的线条检测器代码。像大多数视觉算法一样，Hough变换使用一些参数，这些参数的最佳值（不幸的是）取决于数据，也就是说，一组参数值在一张图片上效果很好，但在另一张图片上可能不是最好的。通过在测试图像上运行你的代码，你将了解这些参数的作用以及改变它们的值对性能的影响。

作为这项任务的一部分，你要实现的许多算法是OpenCV（开放计算机视觉库）中的函数。除非另有规定，否则你不能在本作业中使用这些函数的调用。然而，你可以将你的输出与这些函数产生的输出进行比较，以确保你在正确的轨道上。

1 说明

1. **诚信与协作：**我们鼓励学生以小组形式工作，但每个学生必须提交自己的作品。如果你以小组形式工作，请在你的报告中写明合作者的名字。代码不应该被分享或复制。除非允许，否则请不要使用外部代码。抄袭是被强烈禁止的，并可能导致本课程的不及格。
2. **尽早开始!** 特别是那些不熟悉Python的人。
3. **问题：**如果你有任何问题，请先在Piazza上查看。其他学生可能也遇到过同样的问题，而且可能已经解决了。如果没有，请在讨论区发表你的问题。教学人员会尽快回复。
4. **写作：**你的文章应该主要由三部分组成：（i）你对理论问题的回答，（ii）每个步骤的结果图像（即houghScript.py的输出），以及（iii）对实验的讨论。请注意，我们不接受你在本作业中的手写扫描件。请用电子方式输入理论问题的答案和实验的讨论。
5. **提交：**你提交的作业应该是一个zip文件，命名为<andrew-id.zip>，由你的文章、你的Python实现（包括任何辅助函数）以及你的实现和结果组成。请确保删除data/和result/文件夹，以及你生成的任何其他临时文件。

你最终上传的文件**必须**按照这个布局排列和命名:

- <AndrewID>.zip
  - <AndrewID>.pdf
  - python/
    - houghScript.py
    - myImageFilter.py
    - myEdgeFilter.py
    - myHoughTransform.py
    - myHoughLines.py
    - any helper functions you need
  - ec/
    - myHoughLineSegments.py
    - ec.py
    - 你自己的图像
    - 你自己的结果

gradescope上的自动评分器将检查你的文件格式是否正确；没有通过自动评分器的作业将被扣10分。你的压缩文件<AndrewID>.zip，应该被上传到Gradescope。请注意，Gradescope的文件大小限制为100MB。

6. **文件路径：**请确保你使用的任何文件路径是相对的，而不是绝对的。不是cv2.imread('/name/Documents/subdirectory/hw1/data/xyz.jpg')而是cv2.imread('./data/xyz.jpg')。

2 理论问题

在你的文章中写下你对以下问题的答案。每个问题应该只需要几行字。特别是，“证明”不需要任何冗长的计算。如果你使用许多行复杂的代数中，那做的事情就太复杂（或错误）了。

Q2.1 使用Hough变换参数化线条 (20 points)

1. 证明如果你使用线性方程 $\rho = x\cos\theta + y\sin\theta$, 每个图像点$(x, y)$在$(\rho, \theta)$ 的Hough空间中产生一个正弦波。将正弦波的振幅和相位与点(x, y) 联系起来。

2. 为什么我们用 (ρ, θ) 而不是用斜率和截距(m, c)对直线进行参数化？ 用 (ρ, θ)表示斜率和截距。
3. 假设图像点 (x, y) 在宽度为W 高度为 H的图像中, 即$x \in [1, W]$, $y \in [1, H]$, ρ的最大绝对值是多少, θ的范围是多少？
4. 对于图像中的点（10，10）以及点（20，20）和（30，30），在Hough空间中绘制相应的正弦波，并直观地看到它们的交点是如何确定直线的。这条线的 (m, c)是多少？请用Python来绘制曲线，并在你的文章中报告结果。

3 实现

我们提供了一个名为houghScript.py的主脚本，它负责从一个目录中读入图像，对Hough变换的各个步骤进行函数调用（你将要实现的函数），并生成显示输出和一些中间步骤的图像。你可以随意修改该脚本，但请注意，TAs在评分时将运行原始的houghScript.py。请确保你的代码与原始脚本正确运行，并生成所需的输出图像。

你在这一部分写的每一个脚本和函数都应该包含在 `python/` directory. 请在你写的内容中包括产生的图像。

Q3.1 卷积 (20 points)

编写一个函数，用给定的卷积滤波器对图像进行卷积。

```
img1 = myImageFilter(img0, h)
```

作为输入，该函数需要一个灰度图像 (`img0`) 和一个存储在矩阵h中的卷积滤波器。该函数的输出应该是一个与 `img0` 相同大小的图像 `img1`, 这是 `img0` 与 `h`卷积的结果。 你可以假设滤波器 `h` 沿两个维度都是奇数大小。你将需要处理图像边缘的边界情况。例如，当你把卷积蒙版放在图像的左上角时，滤波器蒙版的大部分将位于图像之外。一种解决方案是在所有这些位置输出一个零值，更好的做法是对图像进行填充，使位于图像边界外的像素与位于图像内的最近的像素具有相同的强度值

你可以调用NumPy的pad函数来填充数组；阅读关于填充的不同模式的文章。然而，你的代码不能调用convolve、correlate、fftconvolve或任何其他类似的函数。你可以将你的输出与这些函数进行比较和调试。

这个函数应该是矢量化的，也就是说，你应该尽可能地依赖对矢量和矩阵进行操作的数学函数。避免一次一次地插入矩阵元素（或尽量少这样做），因为这将大大降低你的代码速度。在这里可以找到矢量化的例子和意义。

具体来说，尽量减少函数中使用的for循环的数量（两个for循环对于实现快速版本的卷积来说是足够了）。

Q3.2 边缘检测 (20 points)

写一个函数，找出图像中的边缘强度和方向。显示你的函数对讲义中的一幅图像的输出。

```
img1 = myEdgeFilter(img0, sigma)
```

该函数将输入一个灰度图像 (`img0`) 和标量 (`sigma`)。`sigma` 是边缘检测前要使用的高斯平滑核的标准偏差。该函数将输出 `img1`, 即边缘大小的图像。

首先，用你的卷积函数对图像进行平滑处理，用指定的高斯-波斯核。这有助于减少图像中的噪音和虚假的精细边缘。使用 `scipy.signal.gaussian` 来获得高斯滤波器的核。高斯滤波器的大小应取决于 `sigma` (e.g., `hsize = 2 * ceil(3 * sigma) + 1`)。

边缘大小的图像 `img1` 可以从 x 方向和 y 方向的图像梯度计算出来。为了找到图像在 x 方向的梯度imgx , 将平滑的图像与 x 方向的 Sobel 滤波器进行卷积。同样地，通过将平滑后的图像与 y 方向的 Sobel 滤波器进行卷积, 得到 y 方向上的图像梯度imgy。如果需要，你也可以输出 `imgx` 和 `imgy`。

在许多情况下，沿着一个边缘的高梯度区域将是相当厚的。对于发现线条来说，最好的办法是让边缘只有一个像素宽。为了达到这个目的，让你的边缘滤波器实现非最大抑制，也就是说，对于每个像素，沿着梯度方向看两个相邻的像素，如果其中任何一个像素有较大的梯度幅度，那么将中心像素的边缘幅度设置为零。将梯度角映射到4种情况中最接近的一种，即线条几乎在0◦、45◦、90◦和135◦处倾斜。例如，30◦会映射到45◦。梯度角可以通过取 y 方向上的梯度大小与 x 方向上的梯度大小之比的反切来计算。

关于非最大限度抑制的更多细节，请参考本讲义的最后一页。

你的代码不能调用OpenCV的Canny函数，或任何其他类似的函数。你可以使用Canny进行比较和调试。图1中显示了一个样本结果。

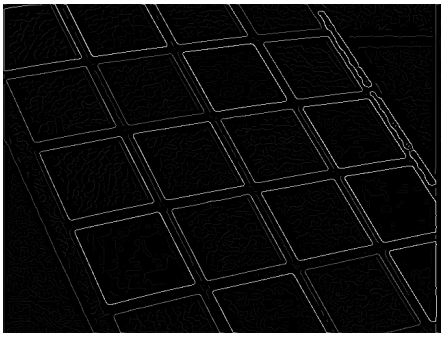

图1：边缘检测结果

Q3.3 Hough变换 (20 points)

编写一个函数，将Hough变换应用于一个边缘明显的图像。在论文中展示输出图像。

`[img_hough, rhoScale, thetaScale] = myHoughTransform(img_threshold, rhoRes, thetaRes)`

`img_threshold` 是边缘大小的图像，经过阈值处理以忽略边缘滤波器响应低的像素。`rhoRes` （标量）是Hough变换累积器的距离分辨率，单位为像素，`thetaRes` （标量）是累积器的角度分辨率，单位为弧度。`img_hough` 是Hough变换累加器，包含所有可能通过图像的线条的"票数"。`rhoScale` 和 `thetaScale` 是 ρ 和 θ 值的数组，`myHoughTransform` 在其中生成了Hough变换矩阵 `img_hough`。例如，如果 `rhoScale[i]` $= \rho_i$, `thetaScale[j]` $= \theta_j$, 那么 `img_hough[i,j]` 包含 $\rho = \rho_i$ 和 $\theta = \theta_j$ 的投票。

超过阈值图像 img_threshold 的每个像素 (x, y) 都是一条线上的可能的点，并在Hough变换中对它可能成为的所有线的一部分进行投票。用 θ 和 ρ 对线条进行参数化，使 $\rho = x\cos\theta + y\sin\theta, \theta \in [0, 2\pi]$, $\rho \in [0, M]$。 M 应该足够大，以容纳可能位于图像中的所有线条。

图像中的每条线都对应于这个范围内的唯一一对 (ρ, θ) 。因此，对应于负 ρ 值的 θ 值是无效的，你不应该计算这些票。

累积器的分辨率需要仔细选择。如果分辨率设置得太低，估计出的行参数可能不准确。如果分辨率太高，运行时间会增加，一条线的投票可能会被分割到阵列中的多个单元。

`img_hough` 的可视化样本见图2。

图2 Hough变换结果

Q3.4 检测线条 (15 points)

编写一个函数，使用Hough变换的输出来检测线条。

```
[rhos, thetas] = myHoughLines(img hough, nLines)
```

其中 `img hough` 是 Hough 变换累积器，`nLines` 是要返回的行数。输出 **rhos** 和 **thetas** 都是 `nLines`×1 的向量，包含 `img hough` 中峰值的行和列坐标，也就是图像中发现的线条。

理想情况下，你希望这个函数能够返回Hough累加器中nLines最高得分单元的 ρ 和 θ 坐标。但是对于累加器中对应于实线的每一个单元格（可能是一个局部的最大值），附近可能会有一些单元格也得到了高分，但不应该被选中。这些非极大值的邻域可以用非极大值抑制法去除。请注意，这个非极大值抑制步骤与之前执行的步骤不同。这里你将考虑一个像素的所有邻居，而不仅仅是沿梯度方向的像素。你可以实现你自己的非最大抑制代码，或者在互联网上找到一个合适的函数(你必须在你的文章中承认并引用来源，同时在你的python/directory)。另一个选择是使用OpenCV函数dilate。一旦你抑制了Hough累积器中的非最大单元，返回累积器中最强的峰值所对应的坐标。

一旦你有了图像中每条线的 ρ 和 θ，脚本 `houghScript.py` 就会根据 ρ 和 θ 的值在图像的边缘画上一条红线。绿色线段代表 OpenCV 的 `HoughLinesP` 的输出（见图3）。

注意，你的代码不能调用 OpenCV 的 `HoughLines`、`HoughLinesP` 或其他类似函数。

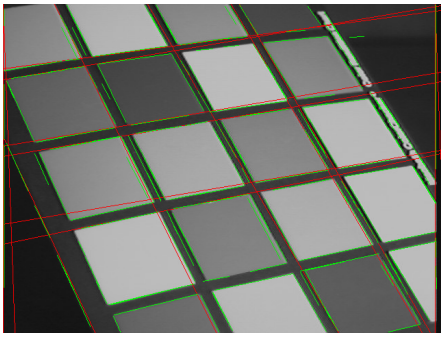

图3：线（红色）和线段（绿色）

Q3.5x 实现自己的*HoughLinesP* (extra: 10 points)

OpenCV 的函数 `HoughLinesP` 将检测到的线条修剪成不超出其所属对象的线段。 现在，轮到我们自己实现一个了！ 请写一个名为 `myHoughLineSegments` 的函数，然后在你的文章中把你的结果与 OpenCV 的函数进行比较。每张图片至少显示一张，并简要描述其差异。

```
lines = myHoughLineSegments(lineRho, lineTheta, Im)
```

你的函数应该将线条输出为一个包含图像中每个线段的起点和终点的像素位置的线条向量。记得将你的实现保存在 `ec/` directory 中。

你的代码不能调用 OpenCV's `HoughLinesP` 函数，或任何其他类似的函数。你可以使用 `HoughLinesP` 进行比较和调试。

4 实验

Q4.1 撰写 (15 points)

使用包含的脚本在图像集上运行你的 Hough 检测器并生成中间输出图像。在你的文章中包括一个图像的中间输出集；这意味着至少包括一个来自 myImageFilter 的图像，一个来自 myEdgeFilter,的图像，以及一个来自 myHoughTransform 的图像。你的代码在所有的图像上都能用一组参数很好地工作吗？最佳参数集是如何随图像变化的？算法的哪一步导致了最多的问题？你是否发现你可以对你的代码或算法进行任何改变，以提高性能？在你的文章中，你应该描述你的代码在不同图像上的工作情况，参数有什么影响，以及你对你的代码做了什么改进以使其工作得更好。请确保花时间编写一份清晰的、完整的记录你工作的文章，以获得满分。

5 非极大值抑制

非极大值抑制 (NMS) 是一种算法，它利用局部极大值的值大于其相邻值的特性来寻找局部最大值。为了在二维图像中实现 NMS ，你可以在图像上移动一个 3×3 (或 7×7, 等等。) 的滤波器。在每个像素上，如果中心像素的值不大于邻居的值（在原始图像中），该滤波器就会抑制中心像素的值（通过将其值设置为0）。如果你发现你的线条非常粗大，这个方法可以用来减薄线条。但是，请注意，它没有考虑到梯度的方向。

要使用 NMS 进行保留线条的边缘减薄，你应该将中心像素的梯度大小与沿梯度方向的邻居进行比较，而不是与所有的邻居进行比较。请记住，梯度是垂直于图像中的线条的（这就是图像梯度的定义）。为了简化实现，你可以将梯度方向量化为 $0°$、$45°$、$90°$ 、和 $135°$中的一个，并根据梯度方向将中心像素与 3×3 窗口中8个邻居中的两个进行比较。如果一个中心像素的梯度大小沿梯度方向小于它的一个或两个邻居的值，那么它的值就被设置为0。

例如，如果一个像素的梯度方向是 $45°$（即图像边缘的方向是 $135°$），我们将其梯度大小与东北和西南的邻居进行比较，如果不大于这两个邻居，就抑制其大小。当一个像素的梯度方向为 $90°$ 时，我们将该像素与它的南北邻居进行比较，如果它的梯度不大于这两个邻居，则抑制其梯度。同样的逻辑适用于梯度方向为 $0°$ 和 $135°$ 的情况。

在实现 NMS 的过程中，你可能会发现 OpenCV 的函数 `dilate` 很有用。