人像识别定位

Portrait recognition positioning

2018011787 核 81 李灵

2021.12

目录

1 问题分析

1.1 问题描述

使用匹配滤波器相关技术在静态图像中进行人像自动定位。基于匹配滤波等相关技术从附件照片中，找到自己的位置。在图中用红色方框自动标出人像所在位置，并显示坐标。

1.2 方法分析

对于基础人像识别定位，可以使用卷积方法逐点匹配得到图像所在位置。[2] 不过该方法运行时间较长，可能需要先对图像进行压缩或是选取其他计算方法提升运行速度。考虑到需要对旋转缩放图像进行定位，不适合选取普通的卷积操作实现。参考文献后发现可以进一步使用 VLfeat[6] 提取 SIFT[9] 特征进行匹配定位，可以在较短的时间内对旋转缩放后的图像得到较好的匹配结果。

2 方法描述

2.1 滤波器

低通滤波器由于对低频信号敏感，因此会保留变化不大的信号，缓和像素值变化明显的地方，也就是通常所说的“平滑”效果；高通滤波器由于对高频信号敏感，因此会放大变化明显的信号，模糊像素值变化不显著的地方，也就是通常所说的“锐化”效果。[10]

2.1.1 滤波方法

本次作业采用低通滤波 [7] 和高斯滤波 [1] 对图像进行处理。

低通滤波是一种低频信号能正常通过，而超过设定临界值的高频信号则被阻隔、减弱的过滤方式。

高斯滤波是对整个图像进行加权平均的操作，即利用每个像素点本身和它周围临近的其他像素点的像素值经过加权平均后得到该点的真正像素值 [1]。它使用正态分布函数计算模糊模板，并使用该模板与原图像做卷积运算，以达到模糊图像的目的。

2.1.2 滤波原理

高斯滤波是一种线性平滑滤波，主要适合于消除高斯噪声。通过读取图像将其转换为灰度图，利用高斯滤波器的滤波核对其进行高斯滤波。此处主要以 SIFT 中的高斯模糊 [11] 为例分析其原理。N 维正态分布方程为：

$$G(r) = \frac{1}{\sqrt{2\pi\sigma^2}^N} e^{\frac{-r^2}{2\sigma^2}} \tag{1}$$

其中，σ 为正态分布标准差，其值越大图像越平滑；r 为模糊半径，指模板元素到模板中心的距离。以二维模板为例，其上元素对应的高斯计算公式 [11] 为：

$$G(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}^N} e^{-\frac{(x-\frac{m}{2})^2 + (y-\frac{n}{2})^2}{2\sigma^2}} \tag{2}$$

二维空间中，原始像素的值有最大的高斯分布值，因此有最大的权重；相邻像素随着距离原始像素越来越远，其权重也越来越小。这样进行模糊处理比其它的均衡模糊滤波器能更好地保留边缘的效果。通常，图像处理程序只需要计算 $(6\sigma+1)\times(6\sigma+1)$ 的矩阵就可以保证相关像素的影响。为了确保模板矩阵中的元素在 [0,1] 之间，需将模板矩阵归一化。[11]

2.1.3 匹配滤波器

结构如图1所示。[8]

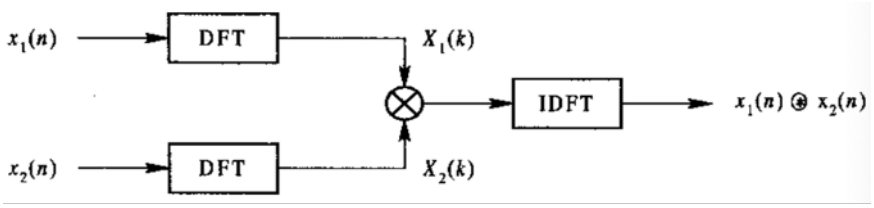


图 1: 匹配滤波器原理示意图

匹配滤波器形如 $h(t)=k*s(t_0-t)$，通过 DFT 将 $h(n)$ 与 $x(n)$ 的卷积变成频域内 $H(k)$ 与 $X(k)$ 的相乘，通过 IDTF 即可得到 $y(n)=h(n)*x(n)$。

2.1.4 滤波结果分析

由于之后会使用 SIFT 算法 [4]，其自带前置高斯滤波功能，因此主要将匹配滤波器用于去除噪声功能的实现。编写函数 denoise，其中 $functionrec=denoise(Img,K0)$，$K0$ 为滤波程度系数。[4] 经过测试，取 $K0$ 为 0.7 时效果较好。将转换后的灰度图图像中的单精度变量储存为 single 类型的 4 字节浮点值方可使用 $denoise$ 函数。

可以看到2，对于左边的图像来说，相当于对其作用了平滑效果；对于有噪点的右边信号而言，则产生了对其去噪的作用。

2.2 卷积算法

图像卷积的本质就是提取图像在不同频段的特征。卷积核本质上是一个二维函数，有它自身对应的频谱函数，因而也可以看作是某种滤波器。通过使用不同的卷积核，可以对图像进行多种不同的滤波操作，得到不同频段的信号。卷积神经网络中的卷积层就是利用了此作用。[10] 由于 DFT 有快速算法 FFT，N 较大时频域内的计算速度会快上许多，因此大多采用 DFT（FFT）计算卷积。

2.2.1 尺度空间

尺度空间使用高斯金字塔进行表示。Tony Lindeberg 指出尺度规范化的 LoG (Laplacion of Gaussian) 算子具有真正的尺度不变性，Lowe 使用高斯差分金字塔近似 LoG 算子 [5]，在尺度空间检测稳定的关键点。其基本思想是在图像信息处理模型中引入一个被视为尺度的参数，通过连续变化尺度参数获得多尺度下的尺度空间表示序列 [4]。对这些序列进行尺度空间主轮廓的提取并以该主轮廓作为一种特征向量，从而实现边缘、角点检测和不同分辨率上的特征提取等。

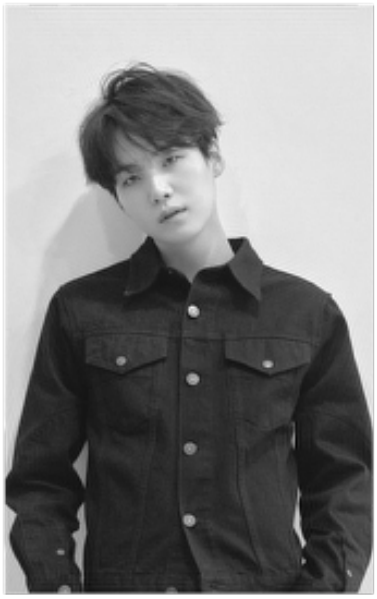


图 2: 去噪后的图像

一个图像的尺度空间 $L(x,y,\sigma)$ 定义为一个变化尺度的高斯函数 $G(x,y,\sigma)$ 与原图像 $I(x,y)$ 的卷积，即：

$$L(x,y,\sigma) = G(x,y,\sigma) * I(x,y) \tag{3}$$

其中，

$$G(x,y,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}^N} e^{-\frac{(x-\frac{m}{2})^2 + (y-\frac{n}{2})^2}{2\sigma^2}} \tag{4}$$

m,n 表示由 $(6\sigma+1)\times(6\sigma+1)$ 确定的高斯模板的维度 [5]。

2.2.2 高斯金字塔

高斯金字塔的构建包括两部分：对图像做不同尺度的高斯模糊和对图像做降采样操作。

金字塔模型是将原始图像不断降阶采样，得到一系列大小不一的图像，再由大到小、由下到上构成一个塔状模型。原图像为金子塔的第一层，每次降采样所得到的新图像为金字塔的一层（每层一张图像），每个金字塔共 n 层。金字塔的层数根据图像的原始大小和塔顶图像的大小共同决定，其计算公式 [5] 如下：

$$n = log_2 min(M,N) - t, t \in [0, log_2 min(M,N)] \tag{5}$$

为保证尺度连续性，高斯金字塔在简单降采样的基础上加上了高斯滤波。Lowe[12] 使用更高效的高斯差分算子代替拉普拉斯算子进行极值检测，公式如下：

$$D(x,y,\sigma) = L(x,y,k\sigma) - L(x,y,\sigma) \tag{6}$$

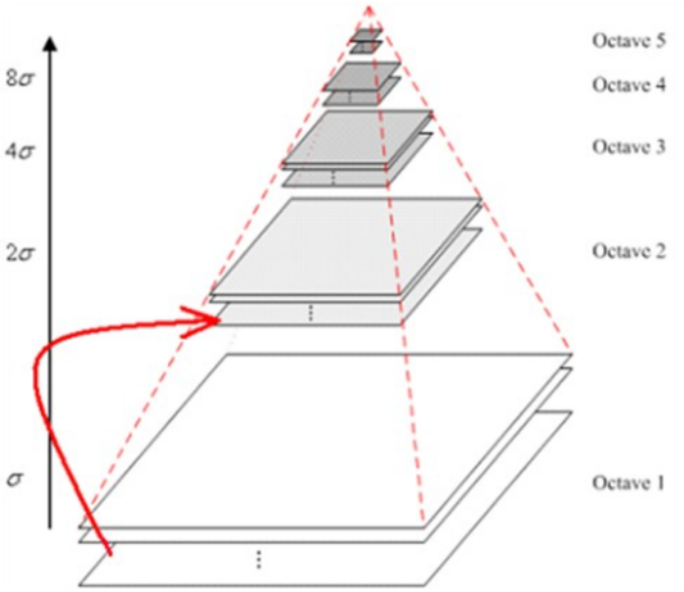


图 3: 高斯金字塔

在实际计算时，使用高斯金字塔每组中相邻上下两层图像相减，得到可进行极值检测的高斯差分图像，如图4所示。

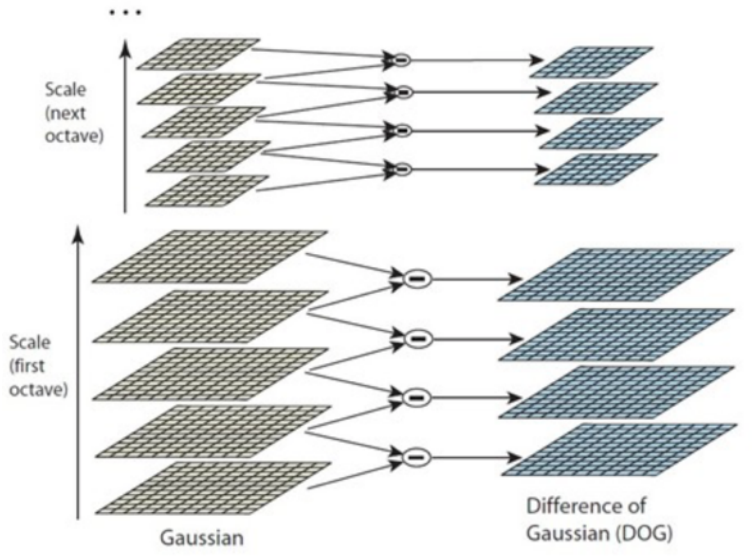


图 4: 高斯差分金字塔的生成

2.2.3 空间极值点检测

关键点是由 DOG 空间的局部极值点组成的，关键点的初步检测是通过同一组内各 DoG 相邻两层图像之间比较完成的。为了寻找 DoG 函数的极值点，每一个像素点要和它所有的相邻点比较，看其是

否比它的图像域和尺度域的相邻点大或者小。如图5所示，中间的检测点和它同尺度的 8 个相邻点和上下相邻尺度对应的 9×2 个点共 26 个点比较，以确保在尺度空间和二维图像空间都检测到极值点。[5]

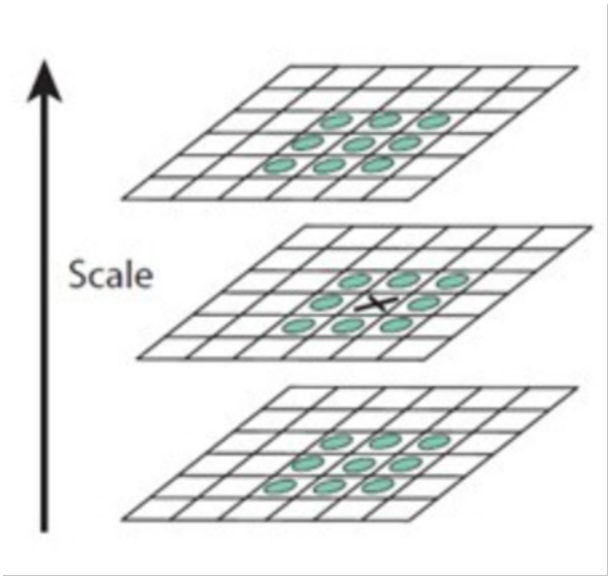


图 5: 空间极值点检测

由此产生的极值点并不一定都是稳定的特征点。某些极值点响应较弱，而且 DOG 算子会产生较强的边缘响应。在构建高斯金字塔时，组内每层的尺度坐标按如下公式计算：

$$\sigma(s) = \sqrt{(k^s\sigma_0)^2 - (k^{s-1}\sigma_0)^2} \tag{7}$$

其中，$\sigma(s)$ 为空间尺度坐标，σ 为组数，S 为组内层数，k 为组内总层数的倒数，即 $k = 2^{\frac{1}{S}}$。Lowe[12] 取初始尺度 $\sigma_0 = 1.6, S = 3$。计算组内某一层图像的尺度时，使用如下公式进行计算：

$$\sigma_{oct}(s) = \sigma_0 2^{\frac{s}{S}}, s \in [0, ..., S+2] \tag{8}$$

考虑到 0 层图像损失，将原图扩大一倍后构建高斯金字塔。

2.2.4 关键点定位

检测到离散空间的极值点后，通过拟合三维二次函数 [3] 来精确确定关键点的位置和尺度，同时去除低对比度的关键点和不稳定的边缘响应点，从而增强匹配稳定性、提高抗噪能力。

（1）提高关键点的稳定性

需要对尺度空间 DoG 函数进行曲线拟合。通过反复插值直到关键点相对于差值中心的距离收敛，对于超出迭代次数或图像边界的点进行删除。

（2）消除边缘效应

通过提取特征点处的 Hessian 矩阵进行比较检验。利用 Lowe 给出的参数可以获得较好的消除边缘响应后的关键点分布。

（3）旋转不变性

使用图像梯度的方法求取局部结构的稳定方向，为给每一个关键点分配一个基准方向以实现旋转不变性。对于在 DOG 金字塔中检测出的关键点，采集其所在高斯金字塔图像 3σ 邻域窗口内像素的梯度和方向分布特征。使用直方图统计邻域内像素的梯度和方向，保留峰值大于主方向峰值 80%的方向作为该关键点的辅方向以增强匹配的鲁棒性。

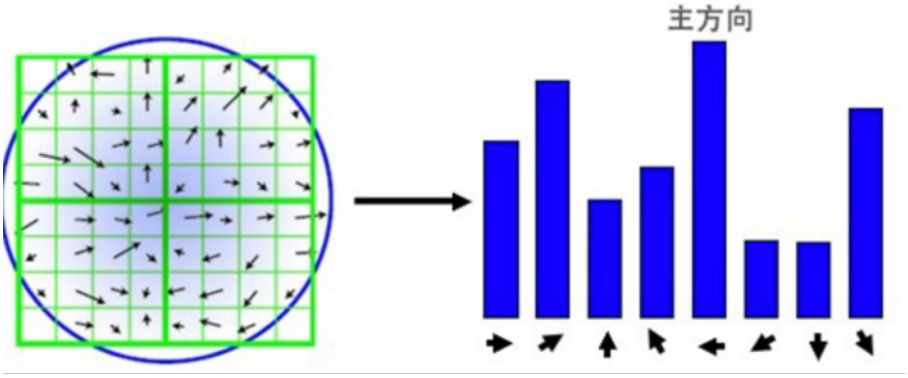


图 6: 关键点方向直方图

把该关键点复制成多份关键点，并将方向值分别赋给这些复制后的关键点，对离散梯度方向直方图进行插值拟合处理，从而求得更精确的方向角度值。

（4）匹配正确度

为每个关键点建立一个包含三个信息（位置、尺度以及方向）的描述符，用一组向量（描述子）将这个关键点描述出来，使其不随各种变化而改变。SIFT 描述子（特征描述向量）是关键点邻域高斯图像梯度统计结果的一种表示。具体实现算法过于繁琐，此处略去。

2.2.5 实现结果

在滤波过程中，已经使用过二维快速傅立叶变换算法进行计算，因此在此不做赘述。对于 SIFT 算法的结果，见 3.2 实验结果。

2.2.6 反思

对于普通卷积计算来说，使用 for 循环所需的时间较长，可采用矩阵计算替代。以及，需要辨别的图像与需要标记的图像大小并不相同，使用平凡卷积较为复杂，可以考虑先对图像大小进行处理或使用高级算法。插值法作为一种数学方法，可以用于提高匹配的精确度。

3 思考改进

3.1 SIFT 算法

3.1.1 简介

尺度不变特征转换 (Scale-invariant feature transform 或 SIFT) 是一种电脑视觉算法，用来侦测与描述影像中的局部性特征。它通过在空间尺度中寻找极值点，并提取出其位置、尺度、旋转不变量而实

现。此算法由 David Lowe 在 1999 年所发表，2004 年完善总结。

3.1.2 算法特点

SIFT 特征是基于图像上的一些局部外观的兴趣点，不仅与影像的大小、旋转无关，对于亮度变化、仿射变换、噪声和视角改变的容忍度也相当高。在母数庞大的特征数据库中，使用 SIFT 特征很容易辨识物体而且很少发生错误。SIFT 特征描述对于部分物体遮蔽的侦测率也相当高，有时只需要 3 个以上的 SIFT 物体特征就足以计算出位置与方位。[5] 在现今电脑的硬件速度和小型特征数据库的条件下，SIFT 特征辨识速度可接近即时运算。

3.1.3 可解决的问题

对进行过下列操作的图像进行特征提取：目标的旋转、缩放、平移；图像仿射、投影变换；光照变化；目标遮挡；杂物场景；噪声叠加。

3.1.4 原理步骤

Lowe 将 SIFT 算法分解为如下四步 [12]：

(1) 尺度空间极值检测：

搜索所有尺度上的图像位置。通过高斯微分函数来识别潜在的对于尺度和旋转不变的关键点。

(2) 关键点定位：

在每个候选的位置上，通过一个拟合精细的模型来确定位置和尺度。关键点的选择依据于它们的稳定程度。

(3) 方向确定：

基于图像局部的梯度方向，分配给每个关键点位置一个或多个方向。所有后面的对图像数据的操作都相对于关键点的方向、尺度和位置进行变换，从而提供对于这些变换的不变性。

(4) 关键点描述：

在每个关键点周围的邻域内，在选定的尺度上测量图像局部的梯度。这些梯度被变换成一种表示，这种表示允许比较大的局部形状的变形和光照变化。

具体实现方式见 2.1.2 滤波原理和 2.2 卷积原理。

3.2 实现结果

无噪声无旋转合成图 运行结果如图7,运行时间约为 25s,输出矩形的四个点坐标为 (754.91, 589.46)(953.87, 589.76)(954.12, 908.99)(755.16, 908.69)。

有噪声无旋转合成图 运行结果如图8,运行时间约为 30s,输出矩形的四个点坐标为 (753.37, 588.38)(954.74, 590.85)(956.13, 909.54)(754.77, 907.07)

无噪声有旋转合成图 运行结果如图9,运行时间约为 25s,输出矩形的四个点坐标为 (900.30, 884.32)(699.66, 850.25)(753.80, 536.16)(954.44, 570.23)

图 7: 无噪声无旋转合成图运行结果

图 8: 有噪声无旋转合成图运行结果

图 9: 无噪声有旋转合成图运行结果

图 10: 有噪声有旋转合成图运行结果


探查摘要(总时间: 25.334 秒)

火焰图

vl_sift | vl_ubcmatch

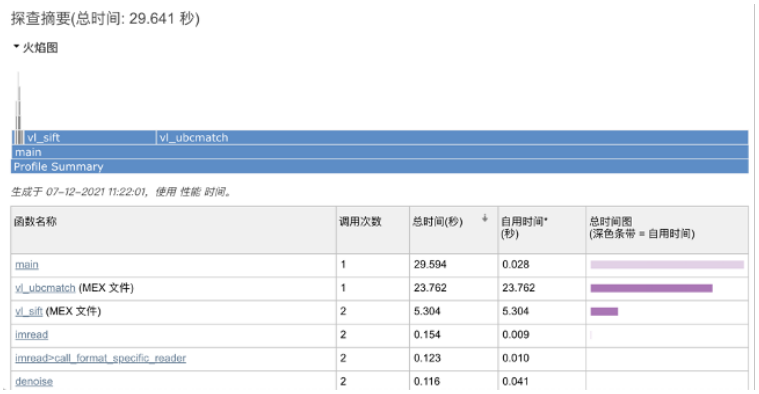
main

Profile Summary

生成于 07-12-2021 11:19:44，使用 性能 时间。

函数名称	调用次数	总时间(秒)	自用时间* (秒)	总时间图 (深色条带 = 自用时间)
main	1	25.277	0.027	
vl_ubcmatch (MEX 文件)	1	19.886	19.886	
vl_sift (MEX 文件)	2	4.811	4.811	
subplot	2	0.122	0.057	
denoise	2	0.121	0.043	
imshow	2	0.114	0.031	

图 11: 无噪声无旋转合成图运行时间


探查摘要(总时间: 29.641 秒)

火焰图

vl_sift | vl_ubcmatch

main

Profile Summary

生成于 07-12-2021 11:22:01，使用 性能 时间。

函数名称	调用次数	总时间(秒)	自用时间* (秒)	总时间图 (深色条带 = 自用时间)
main	1	29.594	0.028	
vl_ubcmatch (MEX 文件)	1	23.762	23.762	
vl_sift (MEX 文件)	2	5.304	5.304	
imread	2	0.154	0.009	
imread>call_format_specific_reader	2	0.123	0.010	
denoise	2	0.116	0.041	

图 12: 有噪声无旋转合成图运行时间


探查摘要(总时间: 24.607 秒)

火焰图

vl_sift | vl_ubcmatch

main

Profile Summary

生成于 07-12-2021 11:28:16，使用 性能 时间。

函数名称	调用次数	总时间(秒)	自用时间* (秒)	总时间图 (深色条带 = 自用时间)
main	1	24.563	0.028	
vl_ubcmatch (MEX 文件)	1	19.274	19.274	
vl_sift (MEX 文件)	2	4.792	4.792	
imread	2	0.135	0.008	
denoise	2	0.118	0.043	
imread>call_format_specific_reader	2	0.092	0.002	

图 13: 无噪声有旋转合成图运行时间

有噪声有旋转合成图 运行结果如图10，运行时间约为 30s，输出矩形的四个点坐标为 (899.46, 896.44)(702.74, 846.35)(755.17, 517.76)(951.90, 567.84)

图 14: 有噪声有旋转合成图运行时间

3.3 反思

对于有无噪声的图像，在处理上还是会出现微小的坐标定位偏差，不过在可接受范围之内。比较运行时间可以看出，对于噪声严重的图像，处理时间也会相应增加。

4 总结

本次作业最开始，我是想简单地使用循环迭代方法进行卷积实现人像识别功能。最初实现的是一个左上角点吻合但大小不匹配的功能，且由于逐点计算，所需时间将近九分钟。尝试降采后发现处理效果大打折扣，于是翻阅了参考文献，决定尝试计算机视觉领域很有名的 SIFT 算法，查阅 Matlab 手册发现已经有人实现了较高质量的 SIFT 算法了，遂下载拓展包采取该方法进行作业。在参考文献 [4] 中，也有人使用 Matlab 代码自行实现了 SIFT 功能。本文主要对其实现的过程进行了描述，重点放在高斯滤波和卷积实现方面，对更多细节的数学推导加以省略。

5 源代码

5.1 main.m

Img1 = imread('assets/photo.png'); Img2 = imread('assets/rotation.tif');
$\text{Img1}_g = single(rgb2gray(Img1)); Img2_g = single(rgb2gray(Img2));$
$\text{Img1}_g = denoise(Img1_g, 0.7); Img2_g = denoise(Img2_g, 0.7);$
[f1, d1] = $\text{vl}_s ift(Img1_g)$; $[f2, d2] = vl_s ift(Img2_g)$;
[matches,] = $\text{vl}_u bcmatch(d1, d2, 2)$;
pos1 = f1(1:2, matches(1, :)); pos2 = f2(1:2, matches(2, :));
[trans, ,] = estimateGeometricTransform(pos1', pos2', 'affine');
[h, w,] = size(Img1); cx = [1 w w 1]; cy = [1 1 h h];
[u, v] = transformPointsForward(trans, cx, cy);
subplot(1,2,1); imshow(uint8(Img1)); hold on;
plot(f1(1,matches(1,:)),f1(2,matches(1,:)),'b*');

subplot(1,2,2); imshow(uint8(Img2)); hold on;
plot(f2(1,matches(2,:)),f2(2,matches(2,:)),'r*');
poly = drawpolygon('Position', vertcat(u, v)');
disp(vertcat(u, v)')

5.2 denoise.m

function rec = denoise(Img, K0) [h, w] = size(Img); ff = fft2(Img);
dx = 1; dy = 1; KX0 = (mod(1/2 + (0:(h-1))/h, 1) - 1/2); KX1 = KX0 * (2*pi/dx); KY0 = (mod(1/2 + (0:(w-1))/w, 1) - 1/2); KY1 = KY0 * (2*pi/dy); [KX,KY] = meshgrid(KX1,KY1);
lpf = (KX.*KX + KY.*KY < K0^2);
rec = ifft2(lpf'.*ff); end

5.3 lpf.m

Img1 = imread('assets/photo.png'); Img2 = imread('assets/rotation$_n$*oise.tif'*);
Img1$_g$ = *single*(*rgb*2*gray*(*Img*1)); *Img*2$_g$ = *single*(*rgb*2*gray*(*Img*2));
Img1$_g$ = *denoise*(*Img*1$_g$, 0.7); *Img*2$_g$ = *denoise*(*Img*2$_g$, 0.7);
subplot(1,2,1); imshow(Img1$_g$, []); *subplot*(1, 2, 2); *imshow*(*Img*2$_g$, []);

参考文献

[1] Matlab 图像处理——图像灰度化，高斯滤波，降采样. https://blog.csdn.net/TJMtaotao/article/details/88915824.

[2] Matlab 简单的人脸识别. https://blog.csdn.net/u014571132/article/details/51569053.

[3] Ransac 算法的 matlab 实现. https://blog.csdn.net/chinaswin/article/details/68066490.

[4] Sift 算法的 matlab 实现. https://www.sun11.me/blog/2016/sift-implementation-in-matlab/.

[5] Sift 算法详解. https://blog.csdn.net/zddblog/article/details/7521424.

[6] Vlfeat. https://www.vlfeat.org/overview/sift.html.

[7] 低通滤波器的 matlab 实现. https://www.mathworks.com/matlabcentral/answers/222864-low-pass-filter-applied-in-frequency-domain-after-fft2-and-before-ifft2.

[8] 匹配滤波器及 matlab 仿真. https://blog.csdn.net/qq_43045275/article/details/106525049.

[9] 基于 sift 特征的图像配准. https://blog.csdn.net/destiny0321/article/details/49341831/.

[10] 通俗理解『卷积』——从傅里叶变换到滤波器. https://zhuanlan.zhihu.com/p/28478034.

[11] 高斯模糊实现小结. https://blog.csdn.net/zddblog/article/details/7450033.

[12] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vis.*, 60(2):91–110, 2004.