频域处理：以傅立叶变换为例

刘朝洋

2016 年 12 月 19 日

专业班级: 计算机 141
学生姓名: 刘朝洋
指导老师: 杨龙

摘要

该实验报告中主要使用 OpenCV 库函数对某个图像进行离散傅立叶变换。另外，为了进一步理解离散傅立叶变换的性质，又分别对几何变换后的图像进行傅里叶变换。通过对比实验结果来验证离散傅立叶变换的性质。

1 实验目的

1. 理解傅里叶变换的原理及方法；
2. 学会使用 OpenCV 对图像进行傅里叶变换；
3. 理解傅里叶变换的几个性质。

2 实验内容

1. 使用 OpenCV 库函数对原图像进行傅里叶变换；
2. 将原图像旋转一定角度再进行傅里叶变换，对比变换结果；
3. 将原图像平移一定单位再进行傅里叶变换，对比变换结果；
4. 将原图像缩小一定比例再进行傅里叶变换，对比变换结果；

3 实验过程

3.1 对图像进行傅里叶变换

下面是对图像进行傅里叶变换的主要步骤：

1. 扩展原图像已获得 DFT 处理的最佳大小；

```
// Expand the image to an optimal size
    int r = getOptimalDFTSize(transformedSrc.rows);
    int c = cvGetOptimalDFTSize(transformedSrc.cols);
    copyMakeBorder(transformedSrc, padded, r - transformedSrc.rows, 0, c -
         transformedSrc.cols, 0, BORDER_CONSTANT);
```

2. 将图像转换为复数形式，并将像素的类型转换为float;

```
// Make place for both the complex and the real values
    Mat planes[] = { Mat_<float>(padded), Mat::zeros(padded.size(), CV_32F) };
    Mat complexSrc;
    merge(planes, 2, complexSrc);
```

3. 调用 OpenCV 库函数对扩展后的图像进行傅里叶变换；

```
// Make the Discrete Fourier Transform
    dft(complexSrc, complexSrc);
```

4. 将傅里叶变换结果转为模长，得到变换的频谱图；

```
// Transform the real and complex values to magnitude
    split(complexSrc, planes);
    Mat mag(transformedSrc.size(), CV_32F);
    magnitude(planes[0], planes[1], mag);
```

5. 为了便于观察，将图像进行对数拉伸和均衡化处理。

```
// Switch to a logarithmic scale
    mag += Scalar::all(1);
    log(mag, mag);
// Normalize
    normalize(mag, mag, 0, 1, CV_MINMAX);
```

6. 为了便于分析和处理，将频谱原点移到图像中心；

```
int cr = mag.rows / 2; int cc = mag.cols / 2;
Mat tl(mag, Rect(0, 0, cc, cr));
Mat tr(mag, Rect(cc, 0, cc, cr));
Mat bl(mag, Rect(0, cr, cc, cr));
Mat br(mag, Rect(cc, cr, cc, cr));

Mat temp;
tl.copyTo(temp);
br.copyTo(tl);
temp.copyTo(br);

tr.copyTo(temp);
bl.copyTo(tr);
temp.copyTo(bl);
```

下面是某图像以及它的 DFT 结果：

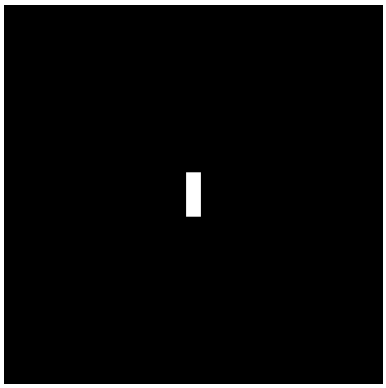

图 1: 原图像

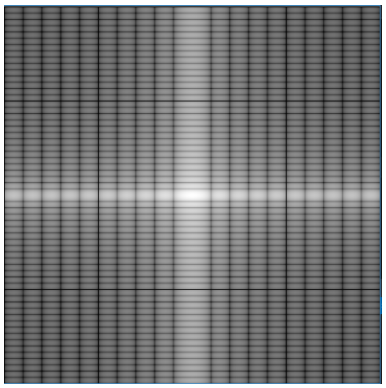

图 2: 原图像 DFT 结果

3.2 DFT 的时移性质

DFT 具有时移性质，即对原始图像进行平移变换后，DFT 的结果仍保持不变，这一性质可用下面的数学表达式表示：

$$f(x - x_0, y - y_0) \Leftrightarrow F(u, v)e^{j2\pi(\frac{ux_0}{M} + \frac{vy_0}{N})} \tag{1}$$

首先我们需要对图像进行平移操作，通过 OpenCV 中的仿射变换 (Affine Transformations) 即可很容易实现。首先要得到变换矩阵，可通过函数`getAffineTransform`

得到，只要向该函数传递变换前与变换后的三个点的位置即可产生该变换矩阵。

```
1    Point2f s[3] = { Point2f(0, 0), Point2f(1, 0), Point2f(1, 1) };
2    Point2f d[3] = { Point2f(x_offset, y_offset), Point2f(1 + x_offset, y_offset),
         Point2f(1 + x_offset, 1 + y_offset) };
3    Mat M(2, 3, CV_32FC1);
4    M = getAffineTransform(s, d);
```

然后将该变换矩阵传递给函数 `warpAffine` 即可。

```
1    warpAffine(src, dst, M, src.size(), 1, 0);
```

下图3为平移变换之后的图像：

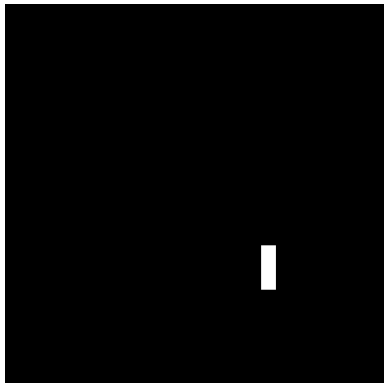

图 3: 平移变换后的图像

最后，使用前面的相同的处理方法，对平移过后的图像做傅里叶变换，下图4是 DFT 结果：

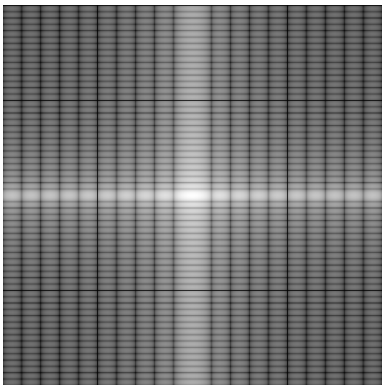

图 4: 平移变换后图像的 DFT 结果

3.3 DFT 的旋转不变性质

离散傅里叶变换也具有旋转不变性质，数学表达式如下：

$$f(r, \theta + \theta_0) \Leftrightarrow F(\rho, \varphi + \theta_0) \tag{2}$$

OpenCV 仿射变换同样可以很容易实现图像的旋转，方法与平移变换类似：

```
1    Point center(src.cols / 2, src.rows / 2);
2    Mat rotMat = getRotationMatrix2D(center, Rotdegree, 1.0);
3    warpAffine(src, dst, rotMat, src.size(), 1, 0);
```

下图5是图像顺时针旋转 90 ° 之后的结果：

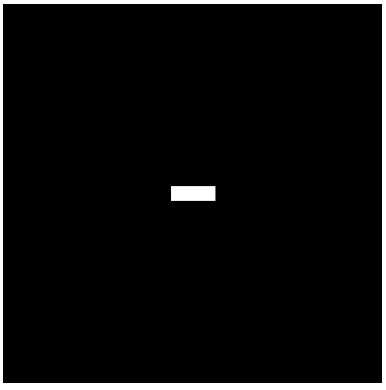

图 5: 旋转变换后的图像

同样，我们需要对旋转后的图像做傅里叶变换，下图6为 DFT 结果：

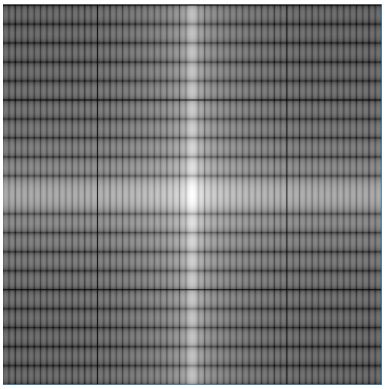

图 6: 旋转变换后图像的 DFT 结果

3.4 DFT 的比例性质

离散傅立叶变换的另一个性质是比例性质，数学表达式为，

$$f(ax, by) \Leftrightarrow \frac{1}{ab} F(\frac{u}{a}, \frac{v}{b}) \tag{3}$$

使用仿射变换同样可以很容易实现图像的缩放。

```
1  Point2f s[3] = { Point2f(src.cols, 0), Point2f(src.cols, src.rows), Point2f(0, src
       .rows) };
2  Point2f d[3] = { Point2f(Scale * src.cols, 0), Point2f(Scale * src.cols, Scale *
       src.rows), Point2f(0, Scale * src.rows) };
3  Mat M(2, 3, CV_32FC1);
4  M = getAffineTransform(s, d);
5  Mat tmp = src.clone();
6  warpAffine(src, tmp, M, src.size(), 1, 0);
7  Mat(tmp, Rect(0, 0, Scale * src.cols, Scale * src.rows)).copyTo(dst);
```

接着对图像进行 DFT，下图7是处理结果：

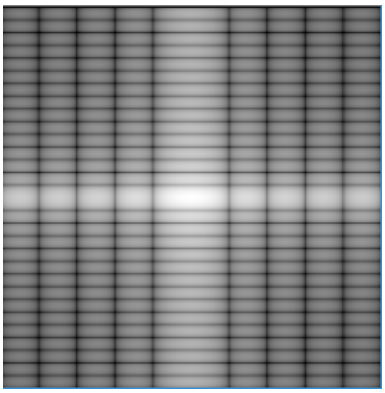

图 7: 缩小后图像的 DFT 结果

4 结果与结论

通过上面的过程，不难发现，无论对图像进行平移、旋转还是缩放，图像的 DFT 结果仍然保持不变。这与离散傅立叶变换的性质，即平移性质、旋转不变性质以及比例性质，完全吻合。