Phase Correlation 을 이용한 이미지 Tracking

박세진 (Se Jin Park, tpwls7423@gm.gist.ac.kr)
광주과학기술원 전기전자컴퓨터공학부

요 약

본 보고서는 Phase correlation 을 이용하여 이미지의 특정 부분을 Tracking 하는 과정을 기술하였다. 동일하게 푸리에 변환, 푸리에 역변환, 푸리에 행렬, Phase Correlation 을 함수로 정의하고 실행하였으며, Activation Map 과 Tracking 된 좌표를 구했다. 언어는 Python 과 Matlab 두가지 언어로 진행하였으며 10 장의 사진에 대해 총 실행시간은 값을 절댓값으로 반환하였을 때 파이썬에서 사진에서 124.5 초 매트랩에서는 104 초 , 값을 실수부만 반환하였을 때의 결과는 파이썬 114 초 매트랩 102 초로 조금 더 실행 시간을 줄이면서 동일한 결과를 얻을 수 있었다..

1. 서론

푸리에 변환은 Continuous, discrete Time 신호에 대해서 시간의 함수에서 주파수의 함수로 바꿔주는 도구이다. 우리는 이러한 푸리에 변환을 통해서 시간 축에서 주파수 영역으로 옮긴 신호를 이용하여 Low/High pass filter 를 사용할 수 있고, 고주파의 신호를 무시해 잡음을 제거하거나, 저주파의 신호를 무시하기도 하며 음성, 사진, 다른 형태의 신호에 대해서 많은 작업들을 수행한다. 본 보고서는 푸리에 변환을 통해 사진을 비교하는 방법 중 Phase Correlation 이라는 위상차 비교 방법을 사용하여, 원래의 사진에서 잘라서 가져온 얼굴 사진 데이터를 활용하여, 다른 사진에서 얼굴을 찾아내는 것을 목표로 하여 진행한다. 개발 환경은 MacbookAir M1 Ram 8Gb 에서 진행하였으며, M1 프로세스 특성상 실행은 Window10 i5-10400, Ram 16Gb 에서 진행하였다.

2. 코드 구성

1)2D Fourier /Inverse Fourier 함수

2D 이미지에 대해서 우리는 1pixel 단위의 Descrete 한 신호에 대해서 Fourier 변환을 하기 때문에 우리는 2D DFT 를 실행하여야 한다.

$$F(u,v) = \sum_{M=0}^{M-1}\sum_{N=0}^{N-1} f(m,n)\exp\left(-2\pi i\left(\frac{mu}{M}+\frac{nv}{N}\right)\right)$$

$$f(m,n) = \frac{1}{MN}\sum_{M=0}^{M-1}\sum_{N=0}^{N-1} F(u,v)\exp\left(2\pi i\left(\frac{mu}{M}+\frac{nv}{N}\right)\right)$$

위의 두 식에 따라서 푸리에 변환과 역 푸리에 변환이 이루러 진다. 이것을 그대로 코드에 옮긴 다면 big-O complexity 를 이용하였을 때, 각각 이미지에 대해서 n^4 의 complexity 를 가지게 되어 비효율적이다. 이에 따라 Fourier Matrix 를 이용하여 compelexity 를 줄였다.

2D 신호의 크기가 $m \times n$이면, fourier matrix 를 이용한 변환은 다음과 같이 이루어 진다.

$$F = FM_m * f * FM_n$$

$$f = FM_m^{-1} * F * FM_n^{-1}$$

각각의 곱은 행렬 곱으로 이루어 지며 FM 은 푸리에 행렬이다.푸리에 행렬의 역행렬은 푸리에 매트릭스의 complex conjugate 에 크기를 나눈 것과 같다. 이에 따라서 n*n 성분을 가진 푸리에 행렬을 구하는 방법은 다음과 같다.

$$FM_n(j,k) = \exp\left(\frac{-2\pi i * j * k}{n}\right), FM_n^{-1} = \frac{1}{n}FM_n^*$$

다음과 같은 푸리에 행렬을 만들고 푸리에 변환, 역변환을 하는 코드는 다음과 같다.

```
function f = Fourier(Mata, Matb, n)
    f = Mata*n*Matb;
end
function f = InFourier(Mata, Matb, n)
    k = size(n);
    f = conj(Mata)*n*conj(Matb)./k(1,1)./k(1,2);
end
function f = Fmat(n)
    for k = 1:n
        for j = 1:n
            f(k,j) = exp(-2*pi*i/n*(k-1)*(j-1));
        end
    end
end
```

Figure 1 Matlab Fourier/Inverse Fourier function

```
def Fmat(n):
    a = np.zeros((n,n),complex)
    for i in range(n):
        for j in range(n):
            a[i][j]=complex(math.cos(2*i*j*math.pi/n),-complex(0,1)*math.sin(2*i*j*math.pi/n))
    return a

def InMat(matrix):
    return matrix.conjugate()/len(matrix)

def Fourier(Mata,Matb,sample):
    i = np.matmul(Mata,sample)
    j = np.matmul(i,Matb)
    return j
def InFourier(Mata,Matb,sample):
    i = np.matmul(InMat(Mata),sample)
    j = np.matmul(i,InMat(Matb))
    return i
```

Figure 2 Python Fourier/Inverse Fourier function

푸리에 행렬을 만들어 주는 Fmat 함수도 같이 정의 하였고 만들어진 이미지 사이즈에 대해서 만들어진 푸리에 행렬들과 이미지 데이터을 인풋 값으로 하여 푸리에/역푸리에 변환을 실행하게 된다.

2) Correlation 함수

Phase correlation 은 두 사진을 비교하는 방법 중 하나로 두 사진의 위상차를 비교하는 방법이다. 우리는 이러한 Phase Correlation 을 사용하여, 우리가 Tracking 하고자하는 이미지를 1 픽셀 씩 이동시키며 Correlation 값이 가장 큰 곳을 찾아내는 방법을 사용한다. Phase Correlation 은 다음과 같은 식으로 사용하였다.

$$R = \frac{\{F(Im_{want}) * F(Im_{find(m,n)})^{*}\}}{\{abs(F(Im_{want}) * F(Im_{find(m,n)})^{*})\}}$$

$$r = F^{-1}\{R\}$$

$$r(m,n) = \max(abs(r)) \; or \; \max(real(r))$$

Im_want 는 우리가 찾고자 하는 이미지이고, Im_find(m,n)은 찾고자 하는 것을 찾아야 하는 이미지에서 크기를 Im_want 크기로 하여, m,n 픽셀 만큼 이동하였을 때의 사진이다. 이 m,n 은 사진 기준 좌측 최상단의 좌표이고, 찾고자 하는 사진의 크기에서 Im_want 의 크기를 뺀 만큼의 좌표 넓이(len(im)-len(Im_want) ,len(im[0])-len(Im_want))에 대해서 실행 하도록 하였다. 각각의 patch 에 대해서 다음을 실행하였을 때 r 은 Im_want 의 크기만큼의 복소수 행렬로써 나오는데, 그 중 절댓값을 취하여 최대값을 찾는 방법과, 실수부 만 취하여 절대값을 찾는 방법 두가지를 설정 하였다. 각각 좌표에 대하여 이러한 r 값이 설정되고, 우리는 그러한 값들 중 최댓값을 찾아 냄으로써 찾고자 하는 이미지(예로 얼굴)을 tracking 할 수 있게 된다. 이러한 것을 코드로 옮기면, 다음과 같다.

```
function f= Corr(Mata, Matb, samplef, data)
    ma = samplef.*conj(Fourier(Mata, Matb, data));
    R = ma./abs(ma);
    r = InFourier(Mata, Matb, R);
    f = max(max(real(r))); %실수부를 반환
    %f = max(max(abs(r))); %절댓값을 반환
end
```

ㅋㅋ

Figure 3 Matlab Correlation function

```
def Corr(samplef, data, Mata, Matb):
    ma = samplef * Fourier(Mata,Matb,data).conjugate()
    R = ma/np.abs(ma)
    r = InFourier(Mata,Matb,R)
    return np.amax(r.real)
    # return  np.amax(np.abs(r))
```

Figure 4 Python Correlation function

인풋 값은 찾고자 하는 sample 을 푸리에 변환시킨 값과, 비교하고자 하는 이미지인 data, 푸리에 행렬들로 구성 되며, return 값은 r 에서의 최대 실수부 혹은 최대 절댓값을 사용하였다.

3)본체 구성

Sample 이미지에 대해서 설명 하자면, 먼저 우리는 찾고자 하는 얼굴 사진을 start.jpg 에서 잘라내게 된다. Start 이미지를 불러와 gray scale 로 1 차원으로 변환시킨 후 잘라냈으며, 잘라낼 좌표의 최상단 위치를 (xpath,ypath), 크기를 dealtax × deltay 로 설정하였다. 각각의 값은 다음과 같다.

```
deltax = 30;deltay=27
deltay = 27;deltax=30
xpath = 275;xpath = 274
ypath = 137;ypath = 136
```

Figure 5 좌표와 이미지 크기

Matlab 은 1 부터 인덱스가 시작하고 python 은 인덱스가 0 부터 시작하기 때문에 차이가 난 것이므로 각각 코드에서 sample 이미지의 좌표와 크기는 동일하다. 이러한 얼굴을 다른 이미지에서 찾기위해 correlation 시키면서 밀어볼 것이므로, 계속 sample 을 fourier 한 값을 쓰게 된다. 그렇게 되면 fourier 를 계속 호출 함으로써 걸리는 시간이 생김으로, sample 을 fourier transform 해준 samplef 를 corr 함수에 넣어서 사용하여 fourier function 의 호출 시간을 줄여준다.

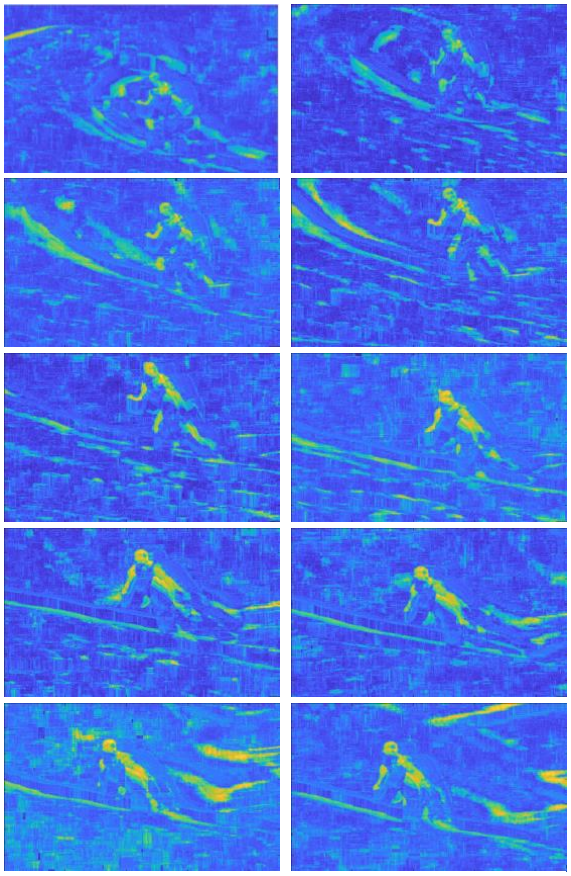

Figure 6 Activation Map

우리는 sample 과 가장 유사한 이미지를 찾는 과정을 진행할 것이다. 이러한 과정은 모든 이미지를 불러와 RGB scale 에서 gray scale 로 만든 다음, 좌측 상단부터 1pixel 씩 이동시키며 correlation 한 값을 얻고 해당 좌표에 할당하여 값을 저장한 후, 할당되어 있는 값들 중 제일 큰 값을 찾는 방법으로 진행 할 것이다. 또한 행렬을 만들어 할당하지 않고 if 문을 통하여 correlation 된 값의 최댓값의 좌표만 찾는 방식으로도 진행하였다. 둘의 complexity 는 동일하다. 하지만 전자는 activation map 을 만들 수 있지만 그만큼 공간이 할애 된다는 점이 있고, 후자는 공간은 아낄 수 있지만 activation map 은 만들지 못한다는 점이 있다. 코드는 다음과 같다.

```
datai = data(:,:,z);
for x = 1:480-deltax
    for y = 1:360-deltay
        k(y,x) = Corr(Mata,Matb,samplef,datai(y:y+deltay-1,x:x+deltax-1));
    end
end
[j1, i1] = find(k == max(max(k)));[j1,i1]
```

Figure 7 Activation Map 얼굴 찾기

```
k= float('-inf')
for i in range(0,len(pix)-deltay):
    for j in range(0,len(pix[0])-deltax):
        data = pixes[l][i:i+deltay,j:j+deltax]
        if k<Corr(samplef, data, Mata, Matb):
            k=Corr(samplef, data, Mata, Matb)
            ki=i
            kj=j
```

Figure 8 Activation Map 없이 얼굴 찾기

Start 이미지를 불러와 찾고자 하는 이미지인 sample 을 만들고 총 10 개의 이미지에 대하여 1pixel 씩 shift 시키면서 각각의 좌표에 대해서 Correlation 한 값을 구하였고, 그것을 통해 Activation map 을 구하였다. 그리고, 각 이미지에서 그러한 Correlation 값의 최댓값의 위치를 알아내었다. Activation Map 을 보았을 때, 우리가 Tracking 하고자 하는 사람 얼굴 이외에도 높은 값으로 나타나는 곳도 존재하였지만 공통적으로 사람의 몸과, 얼굴 근처에서 높은 값을 나타내는 것을 관찰할 수 있다. 이렇게 나타낸 Activation Map 을 이용하여 Correlation 을 한 값이 최대로 나타내는 지점의 좌표를 찾아보는 과정을 할 것이다. 최종적으로 그러한 좌표가 추적한 얼굴이 된다.

3. 결과

각각의 언어로 작성한 코드들을 실행시켰을 때, correlation 값을 절대값으로 반환하였을 때와 실수부 값으로 반환하는 결과는 차이가 없었고, 로 모두 동일 하였다.

또한 각각의 실행 시간은 10 장을 모두 진행하였을 때 다음과 같았다.

Untitled1	1	106.596 s
Untitled1>Corr	1498500	97.661 s
Untitled1>InFourier	1498500	37.368 s
Untitled1>Fourier	1498501	22.699 s

Figure 9 절댓값 사용 Matlab 실행시간

Untitled1	1	102.148 s
Untitled1>Corr	1498500	93.684 s
Untitled1>InFourier	1498500	35.926 s
Untitled1>Fourier	1498501	21.612 s

Figure 10 실수부 사용 Matlab 실행시간

```
time : 13.135573148727417
time : 26.532706022262573
time : 39.92488718032837
time : 52.79310178756714
time : 64.79898500442505
time : 76.59047436714172
time : 88.61630630493164
time : 100.62617897987366
time : 112.49742889404297
time : 124.45444416999817
time : 124.45444416999817
```

Figure 11 절댓값 사용 Python 실행시간

```
time : 11.810407161712646
time : 23.88809847831726
time : 35.728426456451416
time : 47.56775665283203
time : 58.98022770881653
time : 70.08751583099365
time : 81.12000274658203
time : 92.23227787017822
time : 103.23983240127563
time : 114.4229187965393
time : 114.4229187965393
```

Figure 12 실수부 사용 Python 실행시간

파이썬과 매트랩에서의 실행시간 차이는 이미지 불러오는 시간의 차이와 함수를 읽어드리는 시간에서 차이가 났으며 Fourier/Inverse Fourier Transformation 이 호출되어 동작한 시간은 동일한 것을 확인하였다. 각 사진에 대해서 만든 Activation Map 에서 가장 큰 값의 좌표를 찾은 후 그 좌표에서 찾고자 했던 이미지의 크기 만큼의 사각형을 만들어 주면 Figure 11 과 같은 결과가 나오게 된다. 모두 얼굴을 매우 근접하게 찾아 내었으며, 결과는 실수부 사용, 절댓값 사용, 매트랩 사용, 파이썬 사용과 상관 없이 모두 동일하게 구할 수 있었다.

4. 결론 및 향후 개선 방향

2 차원 사진 이미지를 픽셀 단위의 이산 신호로 받아서 원하는 이미지의 한 부분을 찾아 내는 과

Figure 13 최종 결과

정을 하기 위해서, Phase Correlation 을 사용하기 위해서, 2D Discrete time Fourier Transformation / Inverse Fourier Transformation 을 Fourier Matrix 를 만들어서 실행할 수 있게 만들었고 그것으로 다른 사진 파일에 대해서 찾고자 하는 이미지의 한 부분과 비교하였을 때, 값이 가장 큰 곳을 찾아 내는 작업을 실행하였다. 실행 시간을 매트랩 기준 102 초, 파이썬 기준 114 초를 기록하였으며, 찾고자 했어 얼굴 이미지를 잘 찾아 내었다. 그렇다면 어떻게 Phase Correlation 을 이용하여 이미지를 찾을 수 있었을까? Phase Correlation 이란, 비슷한 이미지에 대한 변화에 대한 변위차를 계산하는 방법이다. 우리는 찾고자 하는 이미지와 다른 이미지 간의 변위차를 계산하는 수단으로 Phase Correlation 을 사용하는 것인데, 우리는 Phase Correlation 의 결과로 두 이미지 사이의 상대적인 변환 이동 값을 얻게 된다. 찾고자 하는 이미지와 유사한 이미지를 Correlation 하면 두 이미지 간의 변위차가 모든 Correlation 을 실행한 값 중 최댓값으로 나타나게 되는 것이다. Figure 6 에서 관찰할 수 있듯, 얼굴 이미지와 유사한 형태를 띄는 곳은 상대적으로 다른곳 보다 높은 값을 나타나게 되고, 무엇보다 다른 이미지에서 얼굴을 비교 이미지로 넣어 Correaltion 한 값이 최댓 값으로 나타 나는 것을 확인 할 수 있다. 이러한 비교 방법은 두 이미지를 주파수 영역으로 옮겨 실행하고, 다시 돌아오기 때문에, 푸리에 변환과 역 푸리에 변환이 Correlation 한번을 할 때 마다 한번씩 쓰이게 된다.

시간을 더 줄이거나 얼굴을 더욱더 정확하게 찾아 내게 하는 방법에는 어떤 것이 있을까? 실행 시간을 줄이는 방법에는 행렬 계산에 있어서 Complexity 를 낮추는 방향으로 시간을 줄일 수 있을 것이라 생각한다. 행렬 n *n 행렬의 계산에 있어서는 직접 하나하나 곱하는 방식은 complexity 가 n^3 이고 행렬 계산 알고리즘인 슈트라센 알고리즘이나 윌리엄스의 알고리즘을 적용한다면 Complexity 가 n^2.807 에서 n^2.3727 까지 줄일 수 있을 것이라 예상할 수 있다. 정확도를 높이는 방법에 있어서는 sample 의 값을 계속 갱신 해주는 방법이 있다. 예를 들어 이미지 1 에서 얼굴을 찾고 난 후 이미지 2 에서의 얼굴을 찾기 위해서, Start 이미지에서 자른 얼굴을 계속 쓰지 않고 이미지 1 에서 찾은 얼굴을 새로운 sample 로 하여 찾는 방법을 실행하였다. 프레임의 차가 작은 만큼 비교하는 이미지 간의 위상 차가 작기 때문에 더욱 쉽게 찾을 수 있을 것이라 예상하였다. 하지만 이 방법은, 새로 갱신된 이미지가 튀게 되면 계속 벗어나버리고 결국 다른 곳을 찾아 버리게 된다.

Figure 14 엇나간 이미지

향후 행렬 복잡도 개선을 통해 시간을 더욱 단축시킬 수 있을 것이고, 탐색에 대해서도 많은 알고리즘을 적용시킨다면, 더욱 프로그램을 개선할 수 있을 것으로 기대된다.

참고자료

[1] Oppenheim, A. V., Willsky, A. S., & Young, I. T. (1983). Signals and systems. Englewood Cliffs, N.J: Prentice-Hall.

[2] Wikipedia. “Phase Colleration”, https://en.wikipedia.org/wiki/Phase_correlation , (2021. 5. 21)

[3] Wikipedia. “DFT Matrix”, https://en.wikipedia.org/wiki/DFT_matrix (2021.5.20)

[3] Schönhage, A., & Strassen, V. (1971). Schnelle multiplikation grosser zahlen. Computing, 7(3), 281-292.

[4] Cormen, T. H., & Cormen, T. H. (2001). Introduction to algorithms. Cambridge, Mass: MIT Press.