Signals and Systems Report

Sunghyun Kang
EECS Concentration
GIST College
Gwangju Institute of Science and Technology
kanghyun51015@gm.gist.ac.kr

June 3, 2021

Abstract

In this experiment, we will be looking for tracking images by using Fourier Transformation and Correlation. Starting from obtaining the part which we are going to track, transform that specific part and the whole images to frequency domain where the similarity of the pictures can be quantified. The Fourier Transform, which handles this transformation will be materialized in here. Collecting all the compared values in frequency domain, and connecting to Reverse Fourier Transformation will let us show the part in the total image which is identical to the particular part. This "compared values" can be obtained by various criteria, and this report covers some of the methods of it. Last but not least, introducing a method for efficient and accurate Correlation will be given in the last section.

I. Introduction

IMAGES in time domain are very visible and easy to recognize in human eye. However, many computers will struggle to interpret those pictures since its characteristics are hard to measure. Thus we need to convert the images that is suitable for computers, and this convert method is Fourier Transform.

Once we understood the computer about the image, we will use the inverse Fourier Transformation method whether it will recover the original one. Since Fourier Transformation is linear[1], The result must be similar, but cannot be exactly the same to the initial one because of the *spectral leakage* occurred by the transformation[2].

By using a correlation methods which will be introduced in later, we will track the particular part(i.e. face) in 10 pictures which are given in the class. For the reason that many methods are introduced in this part, I will adopt two variations and see if the results are same or not.

Last but not least, this report will cover some testing about simplified Temporal Consistency method. As to reveal whether this approach are efficient or precise in signal handling, calculating the runtime and inspecting the outputs will be followed.

The following code and simulation are handled by *MATLAB R2021a* version. These codes are described in Appendix section.

II. Fourier Transformation & Inverse Fourier Transformation

In here, as I have mentioned in Introduction, used *MATLAB R2021a* to design Fourier Transformation. Because *MATLAB* cannot accept zero column or rows in their datatype, it is necessary to implement equation 1.

$$F(u,v) = \sum_{m=1}^{M}\sum_{n=1}^{N} f(m,n)e^{-2\pi i(\frac{(m-1)u}{M}+\frac{(n-1)v}{N})} \tag{1}$$

where:

$u = 1, 2, \cdots, M$
$v = 1, 2, \cdots, N$

Since *MATLAB* datatype is taking a form of matrix, this equation also changed into form of matrix. Using Fourier Transform Matrix, we can take these steps.

Let W_m as a matrix induced by $\sum_{m=1}^{M}$ factor, W_n as $\sum_{n=1}^{N}$ factor, and P matrix as the specific patch we want to transfer(The size of it is (M, N)). Then the equation will be followed[3].

$$W_M = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & a_{2,2} & \cdots & a_{2,M} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & a_{M,2} & \cdots & a_{M,M} \end{bmatrix} \tag{2}$$

$$W_N = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & b_{2,2} & \cdots & a_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & a_{N,2} & \cdots & b_{N,N} \end{bmatrix} \tag{3}$$

where:

$$a_{m,u} = e^{-2\pi i(\frac{(m-1)(u-1)}{M})}$$
$$b_{n,v} = e^{-2\pi i(\frac{(n-1)(v-1)}{N})}$$

Using equation 2, 3, we can derive

$$\mathcal{F} = W_M P W_N \tag{4}$$

where:

$\mathcal{F}$ = Fourier Transformation of P

In contrast, Inverse Fourier Transformation will take a form of

$$f(m,n) = \frac{1}{MN} \sum_{u=1}^{M} \sum_{v=1}^{N} F(u,v) e^{2\pi i(\frac{m(u-1)}{M} + \frac{n(v-1)}{N})} \tag{5}$$

where:

$$m = 1, 2, \cdots, M$$
$$n = 1, 2, \cdots, N$$

Using the same steps, we can say

$$W_M^* = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & a_{2,2}^* & \cdots & a_{2,M}^* \\ \vdots & \vdots & \ddots & \vdots \\ 1 & a_{M,2}^* & \cdots & a_{M,M}^* \end{bmatrix} \tag{6}$$

$$W_N^* = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & b_{2,2}^* & \cdots & b_{2,N}^* \\ \vdots & \vdots & \ddots & \vdots \\ 1 & b_{N,2}^* & \cdots & b_{N,N}^* \end{bmatrix} \tag{7}$$

where:

$${a_{m,u}}^* = e^{2\pi i(\frac{(m-1)(u-1)}{M})}$$
$${b_{n,v}}^* = e^{2\pi i(\frac{(n-1)(v-1)}{N})}$$

Using equation 6, 7, we can derive

$$\mathcal{F}^{-1} = \frac{1}{MN} W_M^* \mathcal{F} W_N^* \tag{8}$$

where:

$\mathcal{F}^{-1}$ = Inverse Fourier Transformation of $\mathcal{F}$

Converting those steps in *MATLAB* code, the following procedures are listed in below:

1. Obtain the image section(or whole image) which is needed to convert. The size of the image will be automatically accessed by *size()* function.
2. Get the matrix W_m, W_n, W_m^*, W_n^* by using *for loop*.
3. Figure out $\mathcal{F}$ and $\mathcal{F}^{-1}$ by doing matrix multiplication.
4. See the results by using *imagesc()* function.

In this experiment, the sample image is Figure 1. This is imported to *MATLAB* as Figure 14, which took [272 138 50 50] as an reference patch.

Figure 1: *The sample image which used in Fourier Transformation*

There are succeeding pictures which are also needed to Transform, but this will be introduced in Correlation chapter.

Then, using Fourier Transformation introduced in (4), shows the result as an activation map in Figure 2.

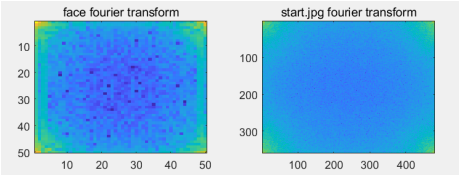

Figure 2: MATLAB *code for solving* $\frac{1}{MN}W_M P W_N$

Doing Inverse Fourier Transformation for the results of Figure 2, it displays an activation map as Figure 3. The code adopted in this process are introduced in Figure 16.

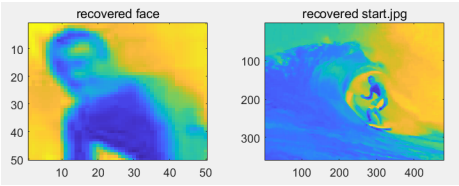

Figure 3: *Inverse Fourier Transformed Figure 5*

As we can see in here, The face and whole image in Figure 1 are well recovered by Inverse Fourier Transform. However, DFT(Discrete Fourier Transform) caused spectral leakage which makes the recovered signals cannot be exactly the same to the original ones.

III. Correlation

Correlation can be achieved by using Fourier Transformation and Inverse Fourier Transformation. This step will be followed in this way:

1. Let total image size as W by H, g_a as reference image patch, g_b as patch extracted in (w,h) location from image which we are searching through, then

$$G_a = \mathcal{F}\{g_a\} \quad G_b = \mathcal{F}\{g_b\} \tag{9}$$

where:

G_a = Fourier Transformation of patch $\{g_a\}$
G_b = Fourier Transformation of patch $\{g_b\}$

2. Then let R as

$$R = \frac{G_a \circ G_b^*}{|G_a \circ G_b^*|} \tag{10}$$

where:

$\circ$ = Element-wise multiplication
G_b^* = Complex conjugate of G_b

3. Set matrix r as

$$r = \mathcal{F}^{-1}\{R\} \tag{11}$$

4. Compare the values inside of r, find maximum value and save it at matrix M which size is W by H. The saving location is g_b's coordinate (w,h).

$$M(w,h) = max(|r|) \tag{12}$$

5. Finally, find maximum value in M and its position. In the image, mark rectangle which height and width is same as reference patch in that location.

$$[row, col] = max(M)'s[row, col] \tag{13}$$

This is designed as a code in figure 17. The result is shown in Figure 4 as an activation map, Figure 5 as an output. As you can see, there are some pitches where the box has missed the face(in img70, img90).

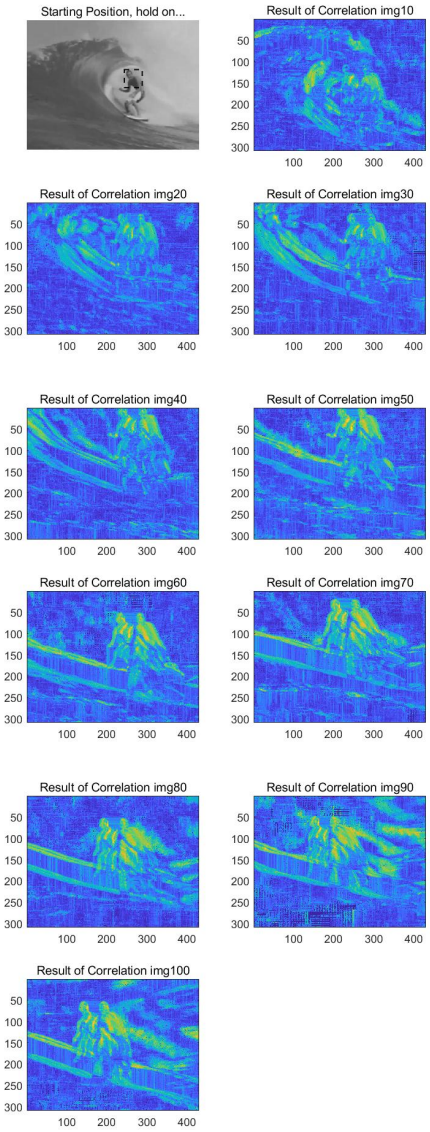

Figure 4: *Activation map for the correlation*

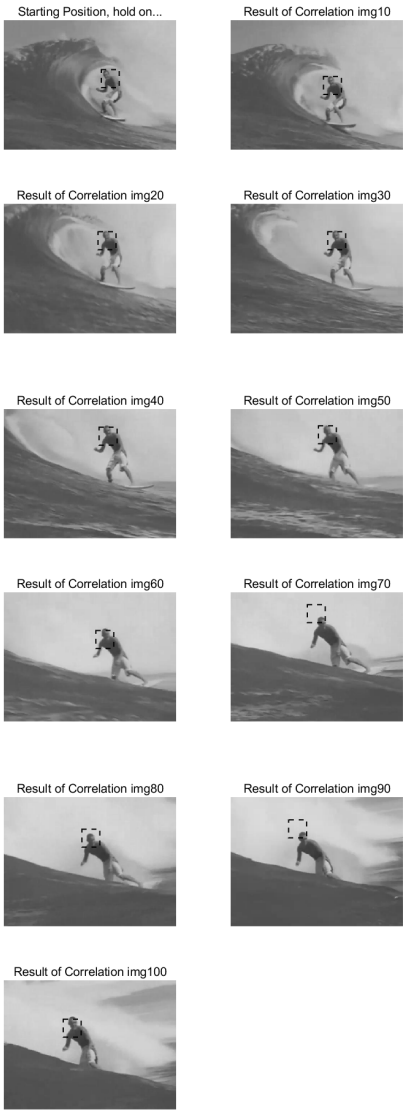

Figure 5: *Correlated Images based on [272 138 50 50]*

Clearly, the image look as "doubled" in Figure 4. This may caused due to the characteristic of this correlation method. In Figure 17 code, the matrix M(in *MATLAB* code, *value* matrix) value gets the correlated value $|r|$ which can be akin to the prior value which M matrix has already taken. Thus it look as same operation has been done twice(but it's not).

As long as the patch was very sensitive at its task, there were numerous trials in finding the right patch that correlates correctly. In conclusion, the reference patch was extracted in [272 138 50 50][1] which is the best case we can managed. Previous trials such as Figure 6 did not shown good outcome. By those trials, there was a tendency that if the reference patch includes both face and armpit, then the accuracy is significantly increased.

The inaccurate cases made by Figure 6 were

[1]This coordinate was produced by the joint research with Huh Gyu(20195191).

described in Figure 7, 8. It shows significant distinction compared to Figure 5.

```
% face: [268 135 39 38]
% armpit: [280 175 20 20]
% face_exactly: [272 138 30 30]
% reshaping1: [270,138,43,48]
% reshaping2: [268,138,45,50]
% goodexcept20: [272 138 45 50]
% BESTEXCEPT70/90(Little): [272 138 50 50]
```

Figure 6: *Correlation coordinates example*

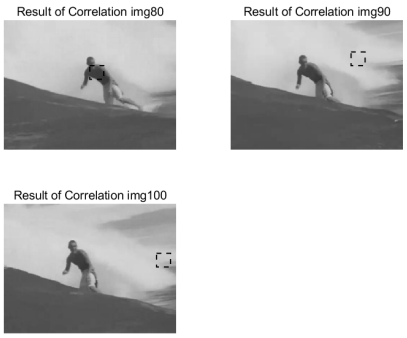

Figure 7: *[268 135 39 38] results*

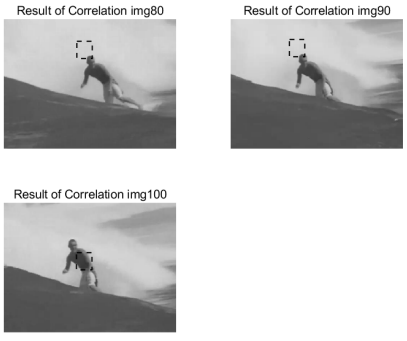

Figure 8: *[270 138 43 48] results*

Also, I used other method to manage correlation based on [272 138 50 50]. which is using $tan^{-1}(\frac{\mathbb{I}\mathrm{m}(\mathrm{r})}{\mathbb{R}\mathrm{e}(\mathrm{r})})$ instead of $|r|$ at equation (12). Then it showed results in Figure 9.

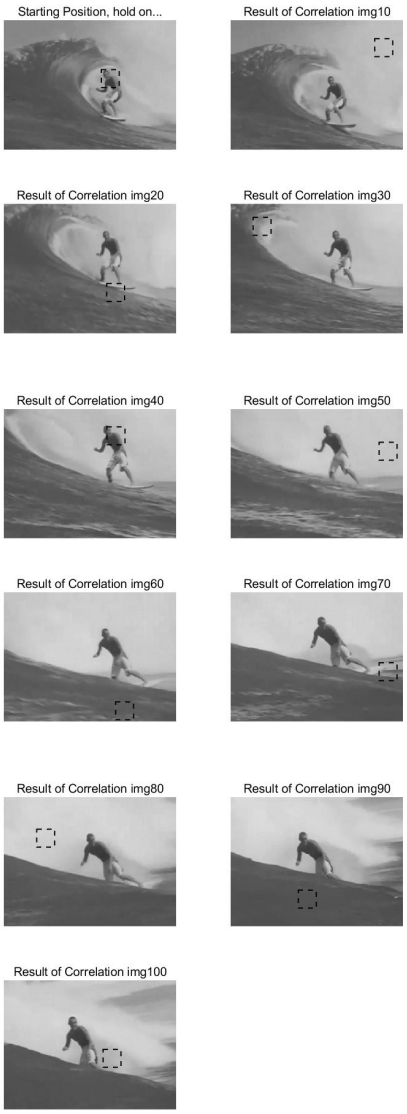

Figure 9: *Phase Correlation with [272 138 50 50]*

It showed devastating results compared to Figure 5. Due to calculation error which *MATLAB* describes in here.

Figure 10: MATLAB *inaccurate calculation error*

IV. Temporal Consistency

To make this Correlation more accurate and quickly in general, I used simplified Temporal Consistency to manage this[4]. It requires these steps:

1. Form start.jpg, extract the stating patch coordinates and send to next correlation.

2. Set specific criteria (e.g. 80 pixels) for the search. Starting from coordinates that given in previous section, explore 80 pixels nearby.

3. Get the result coordinates from this correlation, and send it to next correlation.

4. Repeat until the end.

Utilizing these phase to the code, it is illustrated in Figure 18 and Figure 19.

It shows great power for the ones which showed very inaccurate results like Figure 7. To demonstrate the results and runtime of it, the outcome are described in Table 1 and Figure 11. The runtime was measured by *MATLAB* runtime calculation function *tic toc*, and calculated all correlation(img10 to img100) in measurement. Patch coordinate is [268 135 39 38].

	Runtime	Accuracy
Normal Correlation	779.445 sec	Inaccurate
Temporal Consistency	129.296 sec	More accurate

Table 1: *Runtime and accuracy Comparison in coordinates [268 135 39 38]*

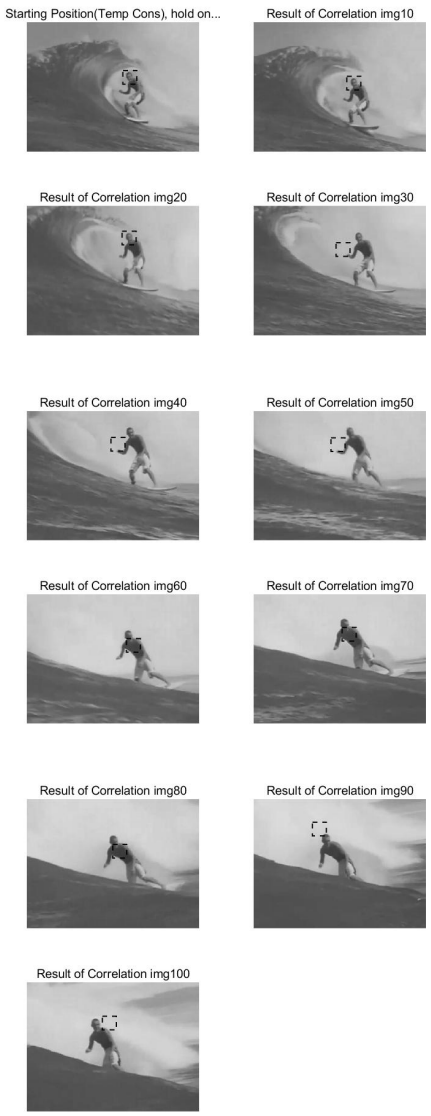

Figure 11: *Image Correlation using Temporal Consistency with coordinates [268 135 39 38]*

Clearly, there are drastic difference in runtime between Normal Correlation and Correlation using Temporal Consistency. Since the detecting area of Temporal Consistency is much smaller compared to Normal Correlation. In addition, we can see improvements in searching the right spot in the middle of the Correlation process compared to Figure 7.

However, once the patch detects wrong location, it results succeeding damaged results as Figure 12. This is caused by vicious circle, that once detects the wrong location as an right spot, then it searches only the nearby in the later picture. This phenomenon can be observed in img70 to img100.

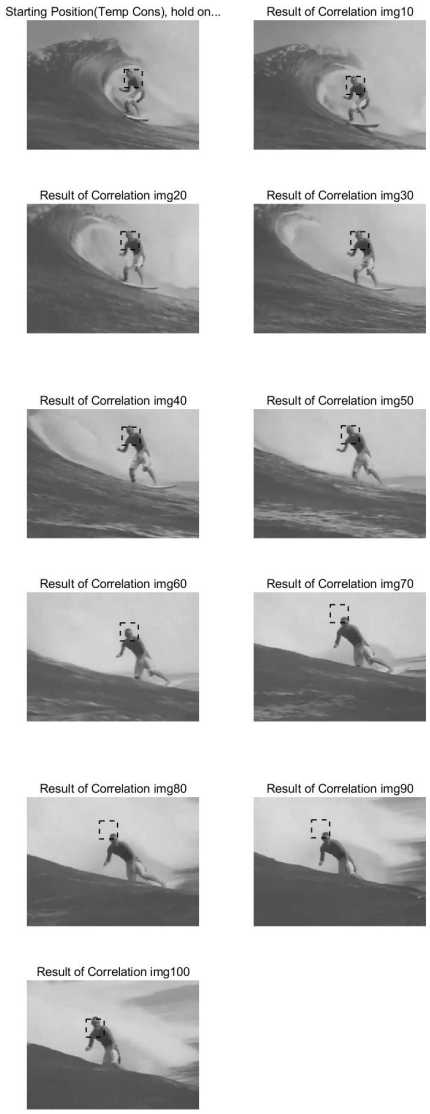

Figure 12: *Image Correlation using Temporal Consistency with coordinates [272 138 50 50]*

V. Conclusion

In visualizing Fourier Transform and Inverse Fourier Transform in the code, we have seen how to use Fourier Matrix in two-dimensional aspect. Because *MATLAB* code shows great power in matrix operations or iteration, it was necessary to make efficient code. At the correlation sector, we can see that the accuracy of correlation is drastically differ by choosing the reference patch in a first place. In here, as introduced [272 138 50 50] patch gave us the best results compared to many examples which was tested previously. Also, changing the correlation criteria(i.e. angle rather than magnitude) makes different correlation results compared to previous correlation, which was very inaccurate. Finally, Temporal Consistency allows us to visualize the Correlation steps more quickly and accurate in common. But constraints are followed due to the lack of error correction.

References

[1] Alan V. Oppenheim, Alan S. Willsky, S. Hamid Nawab. *Signals and Systems*, 2nd edition. New Jersey, US. Prentice-Hall. p. 202.

[2] Harris, Fredric j. *On the use of Windows for Harmonic Analysis with the Discrete Fourier Transform.* IEEE, Jan 1978.

[3] Wolfram Mathworld. *Fourier Matrix*. https://mathworld.wolfram.com/FourierMatrix.html

[4] Debidatta Dwibedi,Yusuf Aytar,Jonathan Tompson,Pierre Sermanet, et al. *Temporal Cycle-Consistency Learning.* CVPR, 2019.

VI. Appendix

```
% Inital Values

f1 = figure;
f2 = figure;
f3 = figure;
f4 = figure;

startimg = imread('C:₩Users₩HyunKang₩Documents₩MATLAB₩source_image₩start.jpg');
startgray = rgb2gray(startimg);
xtart = 272;
ytart = 138;
w = 50;
h = 50;
face = imcrop(startgray, [xtart ytart w-1 h-1]);    % Make image Gray
```

Figure 13: MATLAB *code for getting a image and face patch*

```
function [result] = fourier_transform(patch, M, N)
%   FOURIER_TRANSFORM for total patch
%   M = w, N = h

w_m = (-2*pi)/M;
w_n = (-2*pi)/N;

% for loop for m : M by M Matrix
W_m = ones(M, M);
for m = 2 : M
    for u = 2 : M
        W_m(m, u) = complex(cos(w_m*(m-1)*(u-1)), sin(w_m*(m-1)*(u-1)));
        %W_m(m,u) = exp((w_m*1i*(m-1)*(u-1))*(m-1)*(u-1));
    end
end

% for loop for n : N by N Matrix
W_n = ones(N, N);
for n = 2 : N
    for v = 2 : N
        W_n(n, v) = complex(cos(w_n*(n-1)*(v-1)), sin(w_n*(n-1)*(v-1)));
        %W_n(n,v) = exp((w_n*1i*(n-1)*(v-1))*(n-1)*(v-1));
    end
end
```

Figure 14: MATLAB *code for making* W_m, W_n

```
% Making final fourier transform: Multiplication
% Result is F itself
patch = double(patch);
% patch is always real value
% Hence, multiplication should be done like this:
%result = W_n*patch;
%result = result*W_m;
result = complex(real(W_n)*real(patch), imag(W_n)*real(patch));
result = complex((real(result)*real(W_m)-imag(result)*imag(W_m)),
(real(result)*imag(W_m)+imag(result)*real(W_m)));
end
```

Figure 15: MATLAB *code for solving* $W_M P W_N$

```
function [result] = inverse_fourier(ft, M, N)
% INVERSE_FOURIER_TRANSFORM for total patch
% M = w, N = h

w_m = (2*pi)/M;
w_n = (2*pi)/N;

% for loop for m : M by M Matrix
W_m = ones(M, M);
for m = 2 : M
    for u = 2 : M
        W_m(m, u) = complex(cos(w_m*(m-1)*(u-1)), sin(w_m*(m-1)*(u-1)));
    end
end
W_m = (1/M)*W_m;

% for loop for n : N by N Matrix
W_n= ones(N, N);
for n = 2 : N
    for v = 2 : N
        W_n(n, v) = complex(cos(w_n*(n-1)*(v-1)), sin(w_n*(n-1)*(v-1)));
    end
end
W_n = (1/N)*W_n;

% Making final fourier transform: Multiplication
% Result is F itself
result = W_n*ft;
result = result*W_m;
end
```

Figure 16: MATLAB *code for solving* $\mathcal{F}^{-1}$

```
function Correlation(patch, image)

% Get the size of the patch
p = size(patch);
p = p(2);
q = size(patch);
q = q(1);

value = zeros(480, 360);
for a = 1 : (480-p)
    for b = 1 : (360-q)
        standard = imcrop(image, [a b p-1 q-1]);
        G = fourier_transform(standard, p, q);
        R = (patch.*conj(G))./abs(patch.*conj(G));
        r = inverse_fourier(R, p, q);
        comp = abs(r);
        %comp = atan((imag(r))/(real(r)));
        [findmaxval, ~] = max(comp(:));
        value(a, b) = findmaxval;
    end
end
% Find maximum value's coordinates
[max_val, ~]= max(double(value(:)));
[row, col] = ind2sub(size(value), find(value==max_val));

% Activation Map output, If you want to do get activation map, please use.
%imagesc(transpose(value));

% Image output
% If you want to get activation map, do not run this code.
imshow(image);
hold on
rectangle('Position',[row, col,p,q], 'LineWidth',1 ,'LineStyle','--')
end
```

Figure 17: MATLAB *code for correlation*

```
%-------------------------------------------------------------------------%
% Correlation using Temporal Consistency

% Image searching criteria sr(row), sc(column)
sr = 80;
sc = 80;

tic
figure(f1);
subplot(2, 2, 2), [row, col] = Temporal_Consistency_Correlation(result, img10gray, xtart, ytart, sr, sc);
title("Result of Correlation img10");
figure(f1);
subplot(2, 2, 3), [row, col] = Temporal_Consistency_Correlation(result, img20gray, row, col, sr, sc);
title("Result of Correlation img20");
figure(f1);
subplot(2, 2, 4), [row, col] = Temporal_Consistency_Correlation(result, img30gray, row, col, sr, sc);
title("Result of Correlation img30");
figure(f2);
subplot(2, 2, 1), [row, col] = Temporal_Consistency_Correlation(result, img40gray, row, col, sr, sc);
title("Result of Correlation img40");
figure(f2);
subplot(2, 2, 2), [row, col] = Temporal_Consistency_Correlation(result, img50gray, row, col, sr, sc);
title("Result of Correlation img50");
figure(f2);
subplot(2, 2, 3), [row, col] = Temporal_Consistency_Correlation(result, img60gray, row, col, sr, sc);
title("Result of Correlation img60");
figure(f2);
subplot(2, 2, 4), [row, col] = Temporal_Consistency_Correlation(result, img70gray, row, col, sr, sc);
title("Result of Correlation img70");
figure(f3);
subplot(2, 2, 1), [row, col] = Temporal_Consistency_Correlation(result, img80gray, row, col, sr, sc);
title("Result of Correlation img80");
figure(f3);
subplot(2, 2, 2), [row, col] = Temporal_Consistency_Correlation(result, img90gray, row, col, sr, sc);
title("Result of Correlation img90");
figure(f3);
subplot(2, 2, 3), [row, col] = Temporal_Consistency_Correlation(result, img100gray, row, col, sr, sc);
title("Result of Correlation img100");
toc
```

Figure 18: MATLAB *code for Temporal Consistency, it does sending previous coordinate to the next correlation*

```
function [row, col] = Temporal_Consistency_Correlation(patch,image, inrow, incol, sr, sc)
%TEMPORAL_CONSISTENCY_FUNCTION Uses not only correlation but also temporal
%consistency
% It uses some of the codes inside Correlation.m

% Row Exceptions & Column Exceptions
if inrow - sr < 0
    sr = inrow-1;
end
if incol - sc < 0
    sc = incol-1;
end
if inrow + sr > 360
    sr = 359-inrow;
end
if incol + sc > 480
    sc = 479-incol;
end

% Get the size of the patch
p = size(patch);
p = p(2);
q = size(patch);
q = q(1);

value = zeros(2*sr, 2*sc);
for a = inrow-sr : inrow+sr
    for b = incol-sc : incol+sc
        standard = imcrop(image, [a b p-1 q-1]);
        G = fourier_transform(standard, p, q);
        R = (patch.*conj(G))./abs(patch.*conj(G));
        r = inverse_fourier(R, p, q);
        comp = abs(r);
        [findmaxval, ~] = max(comp(:));
        value(a, b) = findmaxval;
    end
end

% Find maximum value's coordinates
[max_val, ~]= max(double(value(:)));
[row, col] = ind2sub(size(value), find(value==max_val));

% Activation map output
imagesc(transpose(value));

% Image output
imshow(image);
hold on
rectangle('Position',[row,col,p,q], 'LineWidth',1 ,'LineStyle','--')
end
```

Figure 19: MATLAB *code for Temporal Consistency, it does correlation and image drawing*