CSC 420 Assignment 1

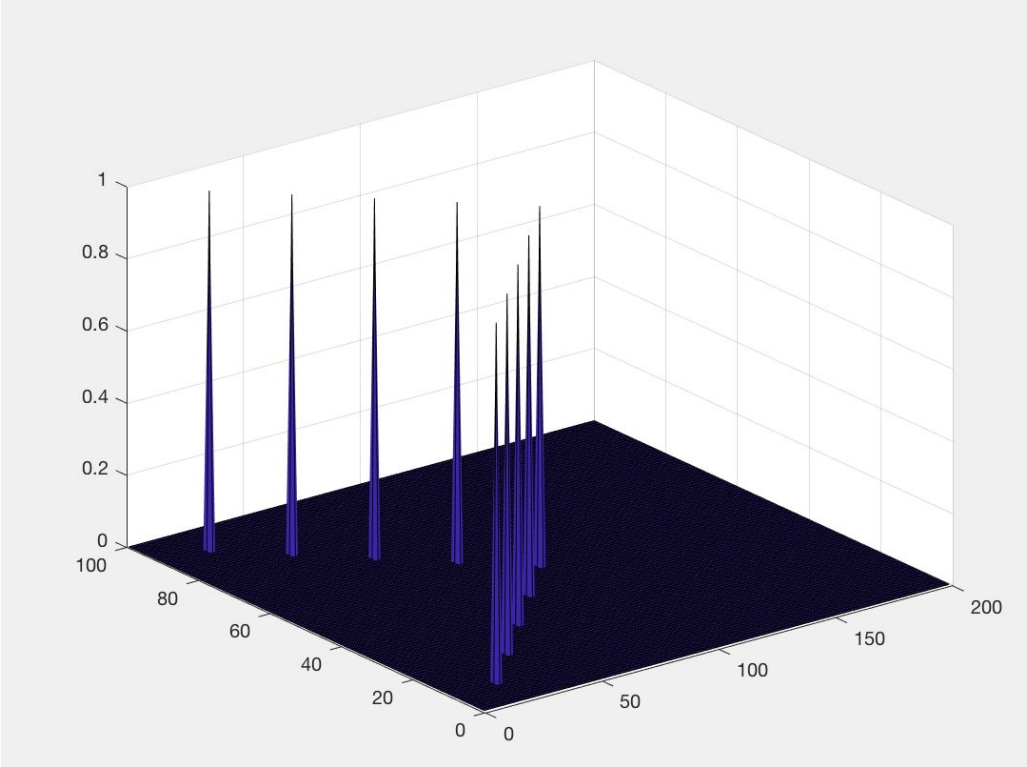
1 a)

$$\delta(u,v) = \begin{cases} 1 \text{ when } u = v = 0 \\ 0 \text{ otherwise} \end{cases}$$

b)

$$\delta(u-m,v-n) = \begin{cases} 1 \text{ when } u = m \text{ and } v = n \\ 0 \text{ otherwise} \end{cases}$$

c)
Plot image:



Code:

```
function impulse()
    Z = zeros(100,200);
    Z(10, 20) = 1;
    Z(20, 40) = 1;
    Z(30, 60) = 1;
    Z(40, 80) = 1;
    Z(50, 100) = 1;
    Z(60, 80) = 1;
    Z(70, 60) = 1;
    Z(80, 40) = 1;
    Z(90, 20) = 1;
    surf(Z);
end
```

d)

$$f(u,v) = \sum_{m=1}^{M} \sum_{n=1}^{N} f(u_m, v_n)\delta(u - u_m, v - v_n)$$

2a)

$$n^2m^2$$

There are n^2 pixels in total, and for each pixels, the filter require m^2 operations. Therefore, the overall runtime would be O(n^2m^2).

b)

$$2n^2m$$

There are n^2 pixels in total, and for each pixels, two filter will be applied. Each filter require m operations. Therefore, the overall runtime would be O(2n^2m) which is proportional to O(n^2m)

c)

F1 is not separable since its rank is equal to 3.

F2 is separable since its rank is equal to 3. And the vectors are the following:

$$\text{row vector} = \begin{bmatrix} 2 & 1 & 2 \end{bmatrix}$$

$$\text{column vector} = \begin{bmatrix} 3 \\ 1 \\ 3 \end{bmatrix}$$

3.a)

This function load the thermometer image as a grayscale double matrix.

```
function thermometer = loadThermometer()
    thermometer = toDoubleAndGreyScale(imread('thermometer.png'));
end
```

```
function grayScale = toDoubleAndGreyScale(template)
grayScale = rgb2gray(im2double(template));
```

b)

This function load each of the templates as a grayscale double matrix.

Then return all the templates and their dimensions as two separate matlab cell.

```
function [templates, dimensions] = readInTemplates
inputFolderRoot = 'DIGITS';
idx = 1 ;
for( s = 1 : 3 )
    inputFolder = fullfile( inputFolderRoot , ['SCALE_', num2str(s)]) ;
    for( i = 0 : 9 )
        templateFile = [ num2str(i), '.png'];
        templates{idx} = toDoubleAndGreyScale(imread( fullfile( ...
        inputFolder , templateFile ) ) );
        dimensions(idx).height = size( templates{idx},1) ;
        dimensions(idx).width  = size( templates{idx},2) ;

        idx = idx + 1 ;
    end
end
```

c)

This function compute correlation between the thermometer and each templates. Then store all the correlations into a M*N*30 matrix. For this part, I did not follow the notation used in the assignment handout. The result is the same since first dimension and second dimension are consistent throughout the code.

```
function correlationArray = constructCorrelationArray(templates, ...
thermometer)
    [M, N] = size(thermometer);
    correlationArray = zeros(M, N, 30);
    for i = 1:30
        correlationArray(:, :, i) = applyCorrelation(templates{i}, ...
            thermometer);
    end
end
function correlation = applyCorrelation(template, img)
    [x, y] = size(template);
    [M, N] = size(img);
    offSetX = round(x/2);
    offSetY = round(y/2);
    fullCorrelation = normxcorr2(template,img);
    correlation = fullCorrelation(offSetX:offSetX+M-1, ...
    offSetY:offSetY+N-1);
end
```

de)

This function finds which template provide best correlation at each pixels. Then prune all the pixels where the best correlation is less than the threshold. Then the x and y value of each of the leftover pixel are recorded.

```
function [maxCorr, maxIdx, candX, candY] = findPotentialNum( ...
    correlationArray, T)
    [maxCorr, maxIdx] = max(correlationArray,[],3);
    [candX, candY] = find(maxCorr > T);
end
```

f)

This function check if each candidate pixel is a local maximum. If so, it print out the bounding box.

```
function finalPruning(ca, maxIdx, candX, candY, ...
dimensions)
    [M, N] = size(candX);
    for i = 1:M
        templateIndex = maxIdx(candX(i), candY(i));
        thisCorr = ca(:,:, templateIndex);
        if isLocalMaximum(thisCorr, candX(i), candY(i))
            drawAndLabelBox(candY(i), candX(i), templateIndex, dimensions);
            drawnow();
        end
    end
end
```

```
function isMax = isLocalMaximum(thisCorr, x, y)
    [M, N, L] = size(thisCorr);
    currentValue = thisCorr(x, y, 1);
    if x > 1 && currentValue < thisCorr(x-1, y, 1)
        isMax = 0;
        return
    end
    if y > 1 && currentValue < thisCorr(x, y-1, 1)
        isMax = 0;
        return
    end
    if x < M && currentValue < thisCorr(x+1, y, 1)
        isMax = 0;
        return
    end
    if y > N && currentValue < thisCorr(x, y+1, 1)
        isMax = 0;
        return
    end
    if x > 1 && y > 1 && currentValue < thisCorr(x-1, y-1, 1)
        isMax = 0;
        return
    end
    if x > 1 && y < N && currentValue < thisCorr(x-1, y+1, 1)
        isMax = 0;
        return
    end
    if x < M && y > 1 && currentValue < thisCorr(x+1, y-1, 1)
        isMax = 0;
        return
    end
    if x < M && y < N && currentValue < thisCorr(x+1, y+1, 1)
        isMax = 0;
        return
    end
    isMax = 1;
    return
end
```

Main function:

```
function main ()
    [templates, dimensions] = readInTemplates();
    thermometer = loadThermometer();
    imshow(thermometer);
    drawnow();
    correlationArray = constructCorrelationArray(templates, thermometer);
    [maxCorr, maxIdx, candX, candY] = findPotentialNum(correlationArray, 0.7);
    finalPruning(correlationArray, maxIdx, candX, candY, dimensions);
```

Final result.
Left one is using provided template which gives 13 correct detections. Right one is using cropped template from input image for some of the numbers which gives 17 correct detections.

By using template from the input images. The correlations can goes up to 1.0 since they are exact matches. This method however also absorb all the noises from the input image. For example, the small 1 at the left bottom corner is covered by dust on the screen. It would not be detected unless we use it as a template. There are also shadow and different lighting affecting the detections which are resolved if we use themselves as templates.