EE112 Final Project: Edge Detection in Images: UPC Decoding

Jay Patel
San Jose State University

Warm-up/ Instructor Verification Form:

2.1(a) Convolve impulses: $\delta[n-3] * \delta[n-5] = \delta[\boldsymbol{n-8}]$

Explanation:

Let x[n] $= \delta[n-3]$, h[n] = $\delta[n-5]$, and y[n] = $\delta[n-3] * \delta[n-5]$

Table 1: Convolution Table to Find y[n]

n	<0	0	1	2	3	4	5	6	7	8	9	10
x[n]	0	0	0	0	1	0	0	0	0	0	0	0
h[n]	0	0	0	0	0	0	1	0	0	0	0	0
h[0]*x[n]	0	0	0	0	0	0	0	0	0	0	0	0
h[1]*x[n-1]	0	0	0	0	0	0	0	0	0	0	0	0
h[2]*x[n-2]	0	0	0	0	0	0	0	0	0	0	0	0
h[3]*x[n-3]	0	0	0	0	0	0	0	0	0	1	0	0
y[n]	0	0	0	0	0	0	0	0	0	**1**	0	0

From the convolution table above, at n = 8, the value of y[n] is 1. Therefore, y[n] = $\delta[n-8]$.

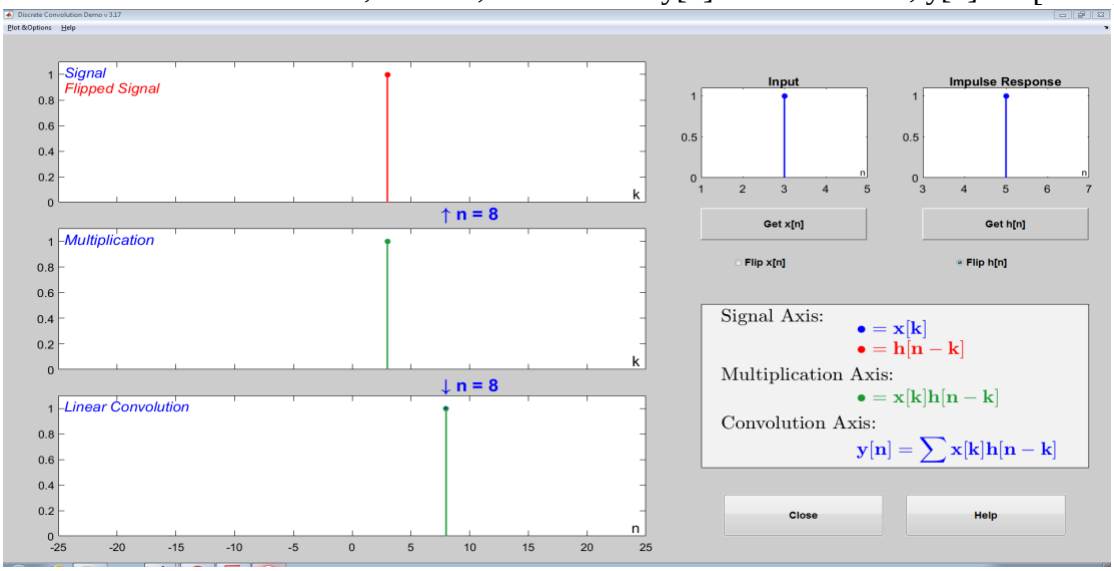

Figure 1a: Convolution of two impulses in in the discrete-time convolution GUI

2.1(b) Rectangular Pulse through a First Difference Filter:

Explanation:

x[n] = (-3)(u[n-2] - u[n-8]) = (-3)u[n-2] + (3) u[n-8]

y[n] = x[n] - x[n-1]

y[n] = (-3){ u[n-2] - u[n-8] } - (-3){ u[n-2-1] - u[-8-1] }

y[n] = (-3) u[n-2] + (3) u[n-8] + (3) u[n-3] + (-3) u[n-9]

y[n] = (-3) u[n-2] + (3) u[n-3] + (3) u[n-8] + (-3) u[n-9]

The stem plot of y[n] is shown in figure 1b below.

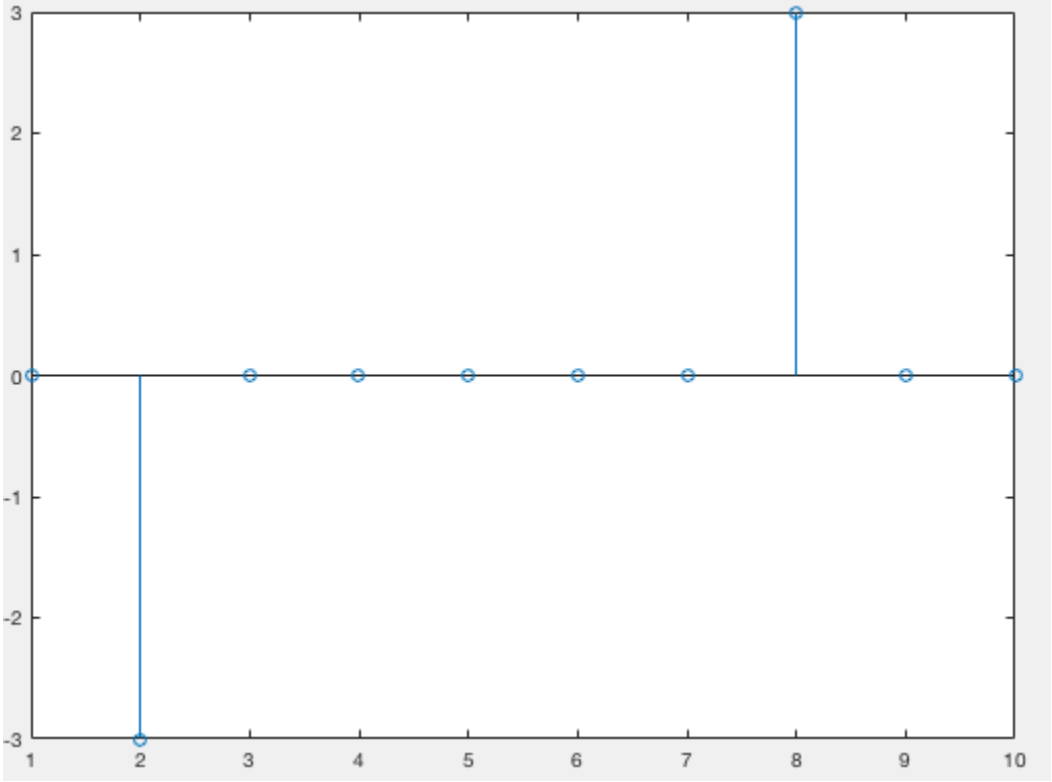

Figure 1b: Stem plot of y[n] = (-3)u[n-2] + (3)u[n-3] + (3)u[n-8] + (-3)u[n-9]

From the figure 1, y[n] can also be written in terms of impulses as shown below,
y[n] = $-3\,\delta[n-2] + 3\,\delta[n-8]$

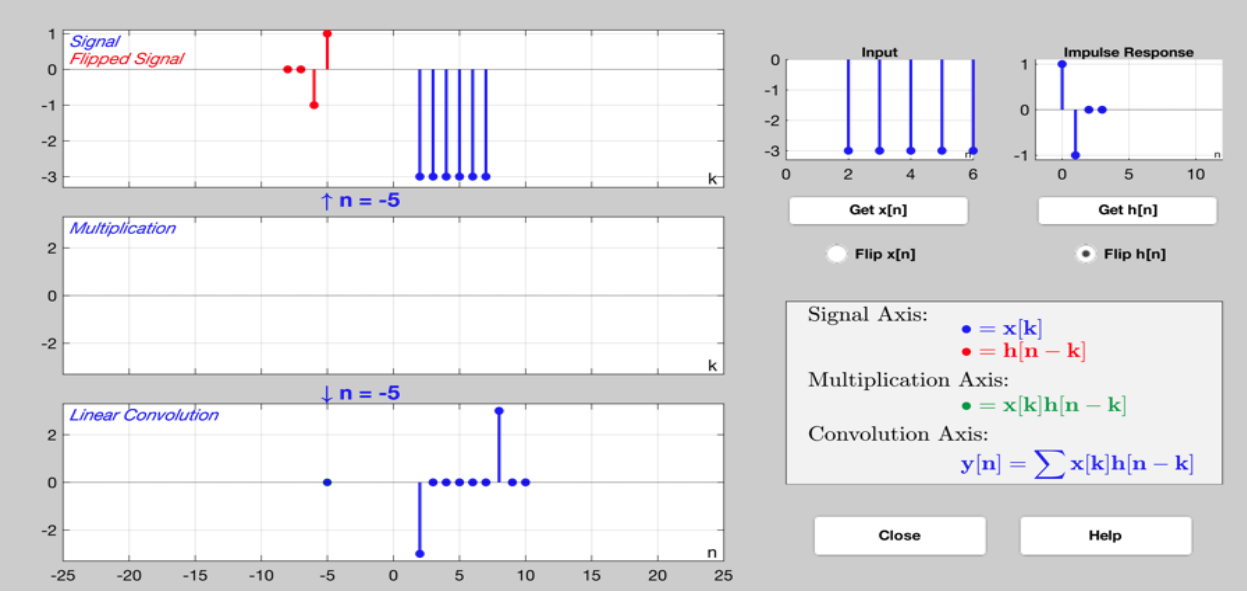

Figure 2: Rectangular Pulse through a First Difference Filter in the discrete-time convolution GUI

2.1(c) Explain why $y[n]$• is zero for most values of n.

Explanation:

As shown in the explanation of 2.1(b), the output of a rectangular pulse through a first difference filter contains four step functions. The first step function has magnitude of (-3) and it starts from n=2. Therefore, all the values of the signal should be (-3) from n=2. However, the second step function has magnitude of (+3), and it starts from n=3. As a result, the total magnitude from n=3 becomes -3 + 3 =0. Therefore, the signal only has a pulse at n=2. Similarly, the step functions with magnitude (+3) and (-3) at n=8 and n=9 respectively, cancels each other except for at n=8. Therefore, there is only one pulse of magnitude (+3) at n=8. As the there are pair of two step functions with equal and opposite magnitude and one time delay, the resulting signal only has two pulses and the rest of the values of the signal are zero.

2.1(d) Convolve two rectangles, sketch result.

Explanation:

x[n] = [2 2 2 2 2 2 2] and h[n] = [3 3 3 3]

y[n] = x[n] * h[n] (Convolution)

The following figure 3, graphically illustrates the convolution of x[n] with h[n] and plots the result.

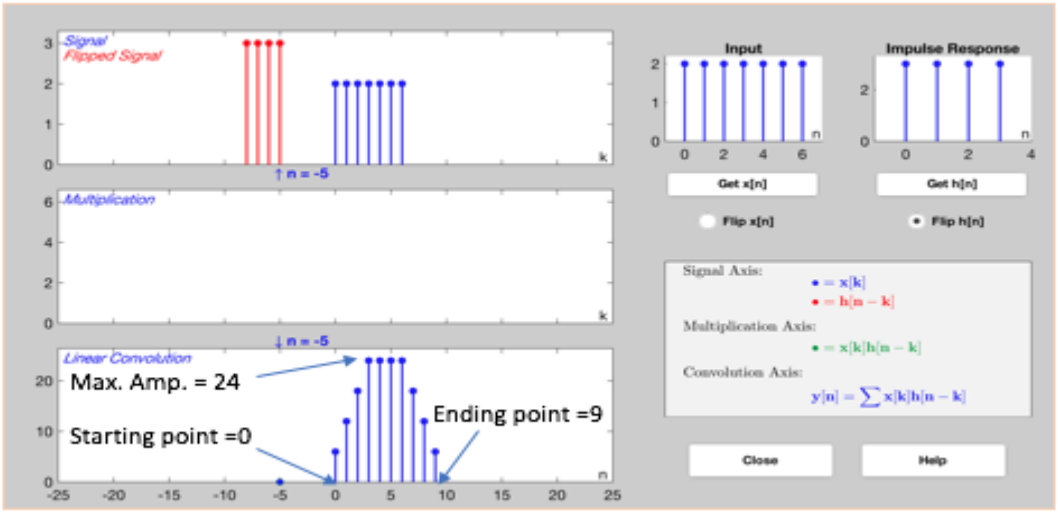

Figure 3: Convolution of two rectangular pulses in the discrete-time convolution GUI

The values of starting point, ending point and the maximum amplitude is pointed out in the figure 3.

2.1(e) Maximum Amplitude and Length of the convolved-rectangles output.
Explanation:
As shown in figure 3, convolution values of two rectangular pulses are,
y[n] = [6 12 18 24 24 24 24 18 12 6]
Therefore,
Length of y[n] = 10
Maximum Amplitude of y[n] = 24

2.1(f) List the index locations (n) of the transitions in the output signal, $y[n]$•
Explanations:
$h[n] = \delta[n] - \delta[n-1]$ = [1 -1]
x[n] = double((sin(0.5*(0:50))+0.2)<0) = [0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 0]
y[n] = x[n] * h[n] (Convolution)
The following figure 4, graphically illustrates the convolution of x[n] with h[n] and plots the result.

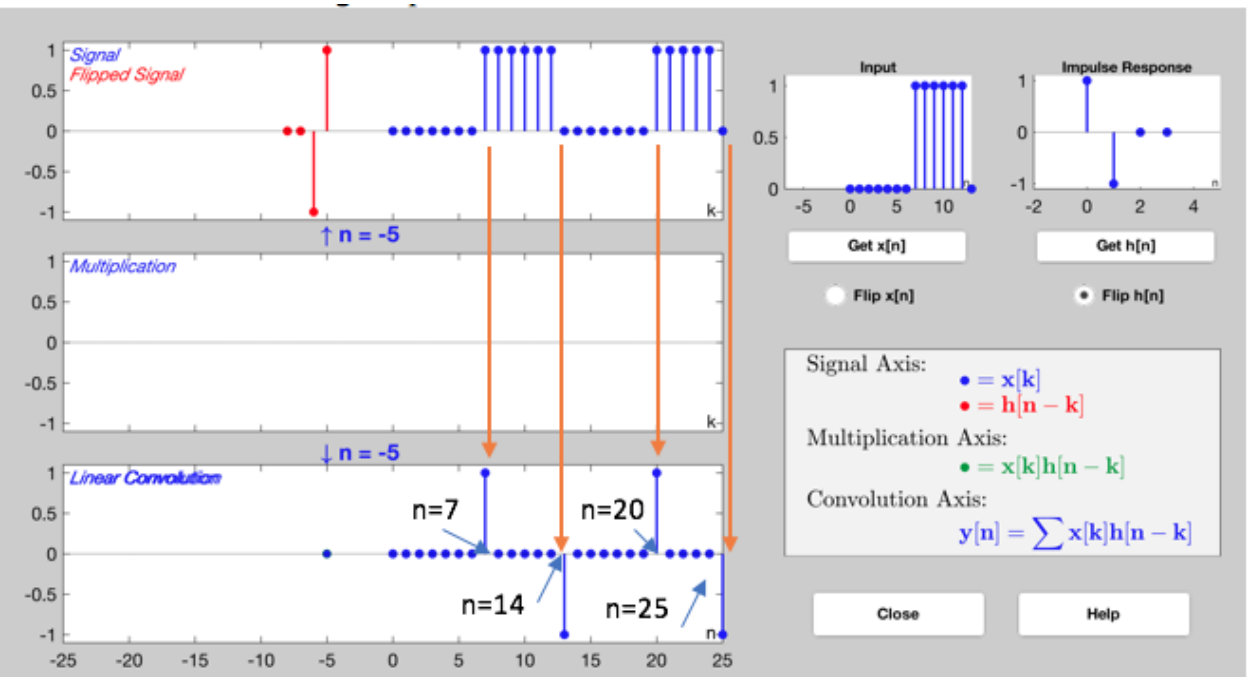

Figure 4: Convolution of x[n] and h[n] in the discrete-time convolution GUI

As seen in figure 4, the index locations of the transition in the output signal y[n] are 7, 13, 20, 25.

2.1(g) Explain polarity (positive/negative) of the transitions in the output signal, $y[n]$
Explanation:
As shown in figure 4, at n = 7, the output y[n] experiences a transition from 0 to 1 as the input x[n] also experiences a transition from 0 to 1. The transitions are shown in the figure 4 by orange arrows. Similarly, at n = 20, the output and input signals shows a positive transition from 0 to 1. For n = 13, the output y[n] experiences a transition negatively from 1 to 0 as the input x[n] also experiences a negative transition from 1 to 0. Same phenomena happens at n = 25. Therefore, the first difference filter h[n] helps generate negative or positive pulses in the output y[n] as x[n] changes negatively or positively respectively.

Part 2.2(a) Process one row of the input image echart with a 1-D first-difference filter. Explain how the output from the filter makes it easy to measure the width of black regions. Use MATLAB's find function to help in determining the width of the black "E" from the impulses in the first-difference output signal.

Explanation:

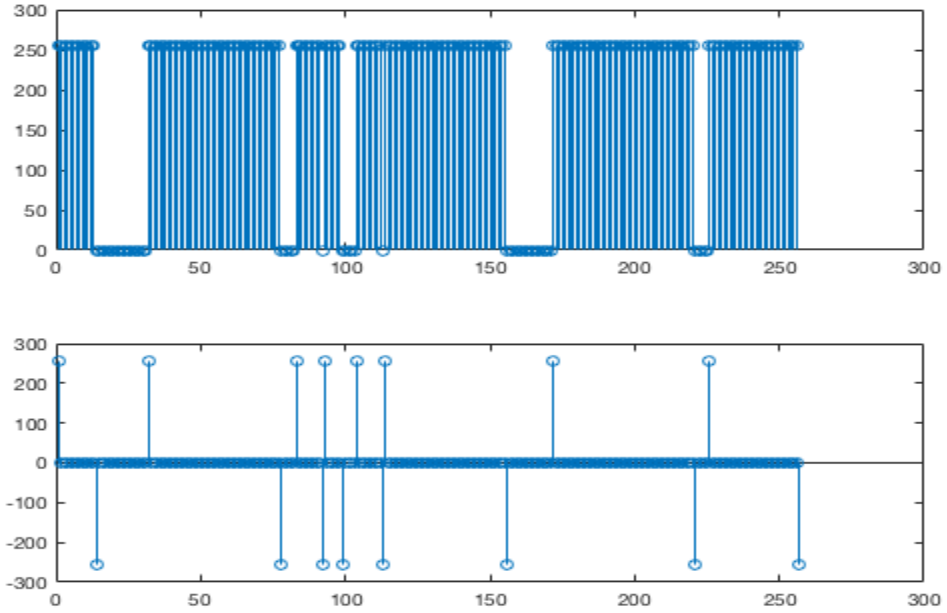

Figure 5: Stem plot of x[n] on the top and y[n] on the bottom

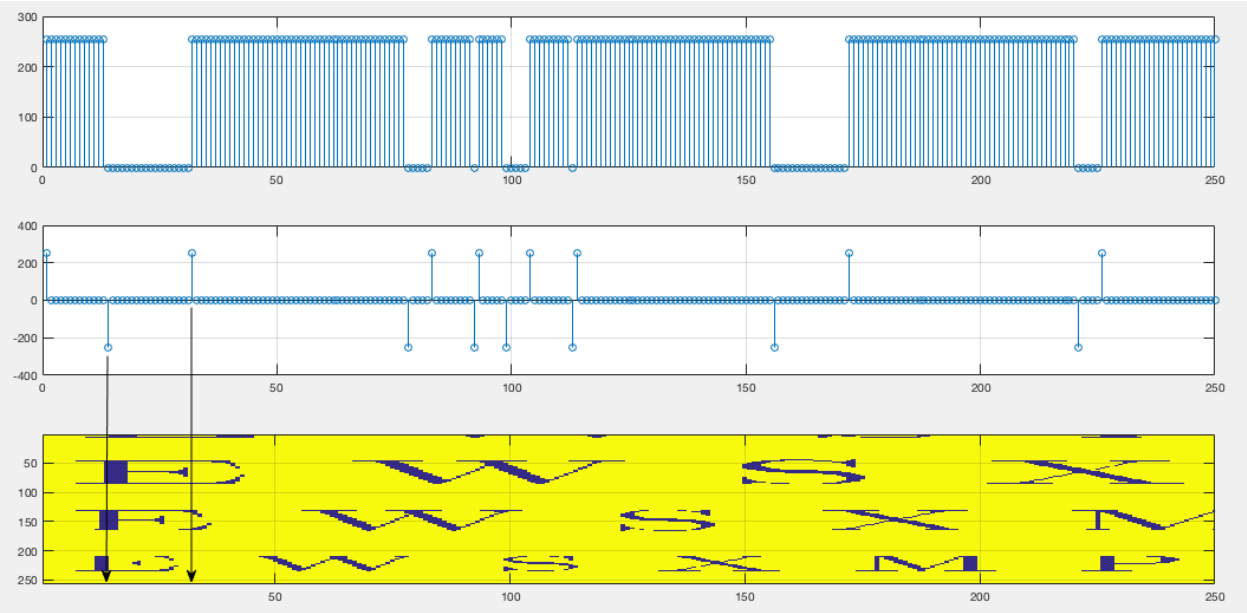

Figure 6: Measuring the width of 'E'

With the difference filter of $x[n] - x[n-1]$ we can detect edges of an image by running a specific row matrix through that difference equation. Intuitively, the difference equation is taking the difference from the current sample with the previous sample -- if a difference exists then the output of that difference filter will not be equal to zero. The width can be quickly measured by virtue of MATLAB's find()function. According to MATLAB, the find() function returns "the

linear indices corresponding to the nonzeros entries of the array…". Simply counting the number of entries between these nonzeros entries yields the width of the object at interest. Figure 5 shows the stem plot of the 65th row of the 257x256 echart image provided for this experiment.

Observing the stem plot of the 65th row of the `echart` shown in Figure 6, it is clear that the letter "E" is 32 - 14 = 18 samples wide.

MatLab Code used in part 2.2:

```
>> load ('echart.mat'); % Load the file echart,mat into echart
varuable.
>> bdiffh = [1 -1] %First order difference Filter
>> m = 65 % Randomly picked row number
>> xx1 = echart (m,:); % The values of row 65 stored in xx1
>> yy1 = conv (echart(m, :), bdiffh) % output y[n] as convolution of
x[n] and difference filter
>> subplot(211), stem (xx1) % stem plot of x[n]
>> subplot(212), stem (yy1) % stem plot of y[n]
>> find(yy1) % Locations of y[n] where there is a transition

ans =

     1    14    32    78    83    92    93    99   104   113   114
156   172   221   226   257
```

3. Edge Detection and Location via 1-D Filters

MatLab Code used in Edge Detection and Location via 1-D Filters

```
>>%part (a) code
>> xx = 255*(rem(1:159,30)>19);
>> bb = [1 -1];
>> yn = conv (xx,bb);
>> subplot(211), stem(xx)
>> subplot(212), stem(yn)
>>%_____________________________________
>> %part (d) code
>> for m = 1:160
       if ( abs(yn(1,m)) > 200) % Threshold value: τ = 200
           dn(1,m) = 1; % Set d[n] to true or 1; edge is detected
here.
```

```
        else
            dn(1,m) = 0; % Set d[n] to false or 0
        end
    end
>>%_____________________________________
>>%part e
>> ln = find (dn);
>> subplot (111), stem (ln)
```

(a) Figure 7 shows the discrete-time signal plot of x[n] and y[n]. The code to plot the figure 7 is shown in the matlab code above.

(b) Simply put, difference FIR Filter of $y[n] = x[n] - x[n-1]$ is outputting the difference of the current sample with that of the previous sample.

Let $x[n] = 255 \cdot (u[n-20] - u[n-30] + u[n-50] - u[n-60] + \cdots)$

and $y[n] = x[n] - x[n-1]$.

Convoluting the two above signals we obtain

$$y[n] * x[n] = 255(\delta[n-20] - \delta[n-30] + \delta[n-50] - \delta[n-60] + \cdots)$$

This make sense -- the difference equation is taking the difference of the current value with the previous value. When the value of 255 is sampled and the previous sampled value remained 0, the difference proved to be 255. However, when *n* is incremented once more, the difference equation samples 255 and 255. The difference of 255 and 255 is 0, which is exactly the behavior seen in Figure 7 and in the above mathematical calculation.

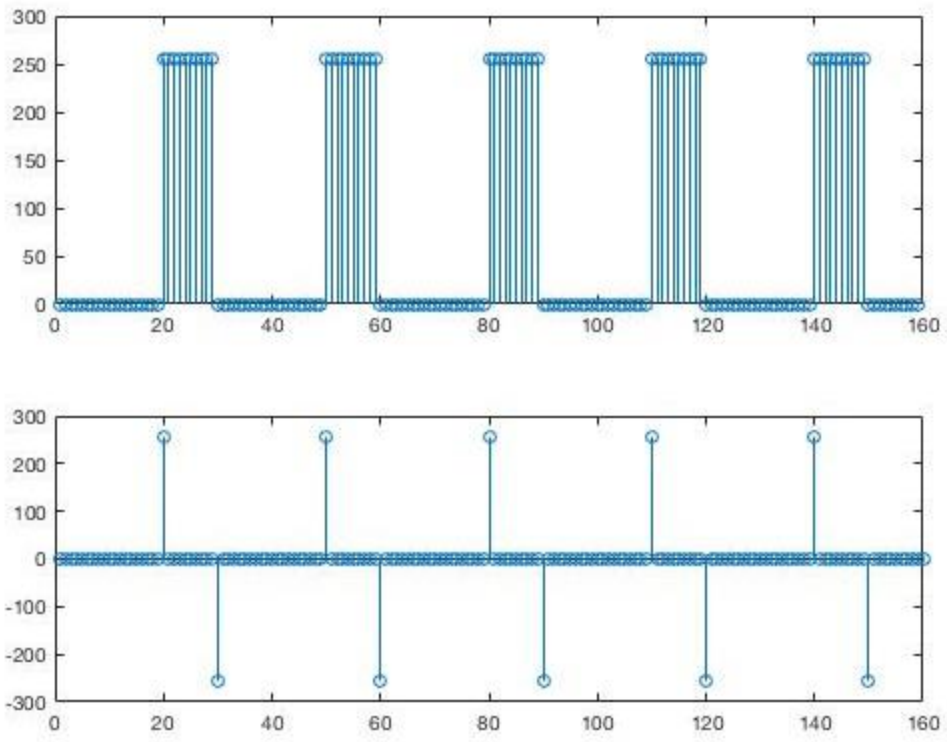

Figure 7: Discrete-time signal plot of x[n](top) and y[n](bottom)

(c) The original signal was 1 by 159 matrix but the signal that came out of the FIR difference filter was 1 by 160. Why is the filtered signal one sample longer? A simple formula for calculating the length of a filtered signal is

$$\textit{Filtered Length} = \textit{Original Length} + \textit{Length of FIR} - 1$$

$$\textit{Filtered Length} = 159 + 2 - 1$$

$$\textit{Filtered Length} = 160$$

Therefore the length of the filtered signal will be one sample longer than the original signal.

(d) The code to convert the signal y[n] to the signal d[n] that contains the values of only zeros and ones where one represents an edge is shown in the matlab code above. The value of threshold is chosen to be 200 as all the edges have the absolute magnitude of 250.

(e) Figure 8 demonstrates discrete-time signal of the edge locations. As there are ten edges in the signal y[n] as shown in figure 7, the length of the signal is ten.
Length of the plotted signal = 10.

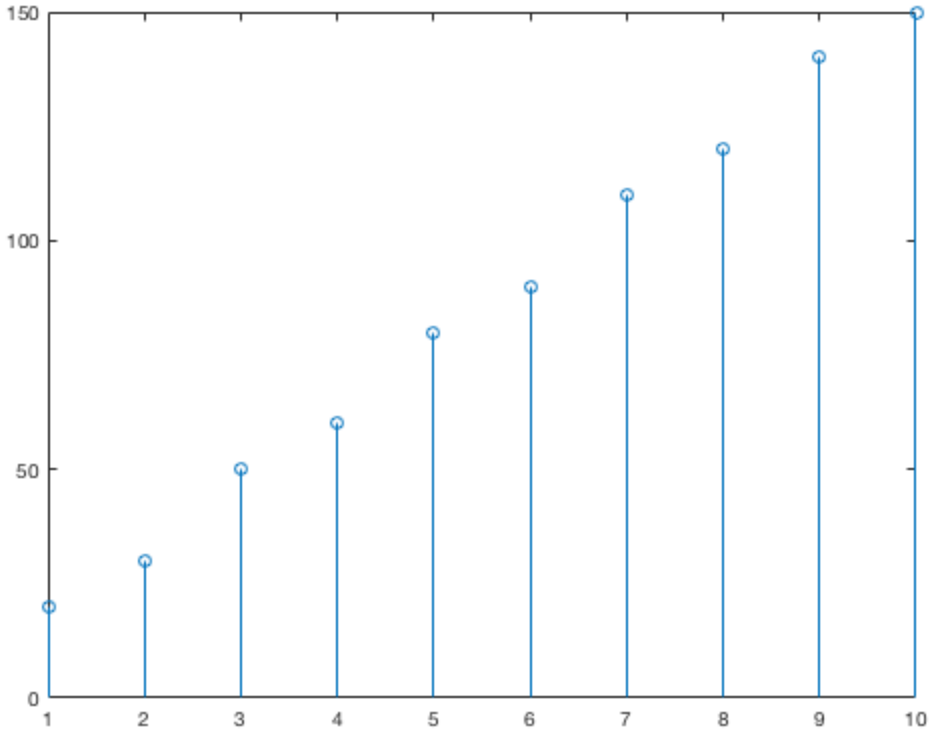

Figure 8: Stem plot of edge locations of y[n]

3.2.1 Decode the UPC from a Scanned Image

(a)

```
MatLab code used in part (a):
>> bcode = imread('HP110v3.jpg');%Barcode Image loading
>> xn = bcode(250, :);% row 250 was chosen randomly to
process >> %the barcode
```

For this part of the experiment, the UPC image shown in Figure 9 was loaded onto MATLAB and the 250th row was extracted and saved onto an 1x949 vectors labeled ‘xn’.

Figure 9: 500 x 948 UPC Image File 'HP110v3.PNG'

(b)

```
MatLab code used in part (b):
>> difh = [1 -1];% difference filter
>> yn = conv (double(xn),double(difh));% Filtered xn
>> subplot(211), stem(xn)
>> subplot(212), stem(yn)
```

The 'xn' vector was the convoluted using the difference equation of $y[n] = x[n] - x[n-1]$ and saved onto a 1x949 vector labeled 'yn'. Next, the stem plot of 'xn' and 'yn' was plotted as shown in Figure 9.

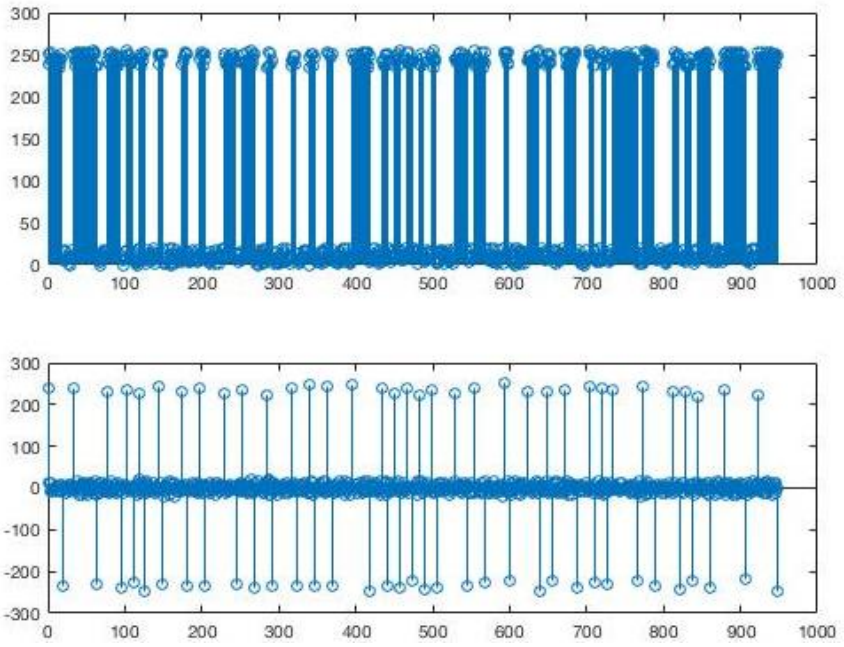

Figure 10: Stem Plot of 'xn' on top and the Filtered Step Plot 'yn' on the bottom

(c)

```
MatLab code used in part (c):
%The following for loop creates a threshold signal with
%zeros and ones where one represents an edge.
>> for m =1:949
      if (abs(yn(1,m)) < 50) % Threshold value: τ = 50
         dn(1,m) = 0;
      else
         dn(1,m) = 1;
end
end
>> ln = find (dn);
>> stem (ln)
```

The threshold value of the signal was set to 50 as the magnitude of the pulses at the edge had the absolute value of about 250. Any pulse that has magnitude of less than 50 has to be ignored since the pulses with less than 50 magnitude did not represent the edge. The signal d[n] represents the threshold signal that contains the values of only zeros and ones where one represents an edge. The locations of the edge, locations of ones in the d[n] signal, occurrence was then saved into the signal l[n] using `find` function. The stem plot of the signal l[n] is showed in figure 11 below.

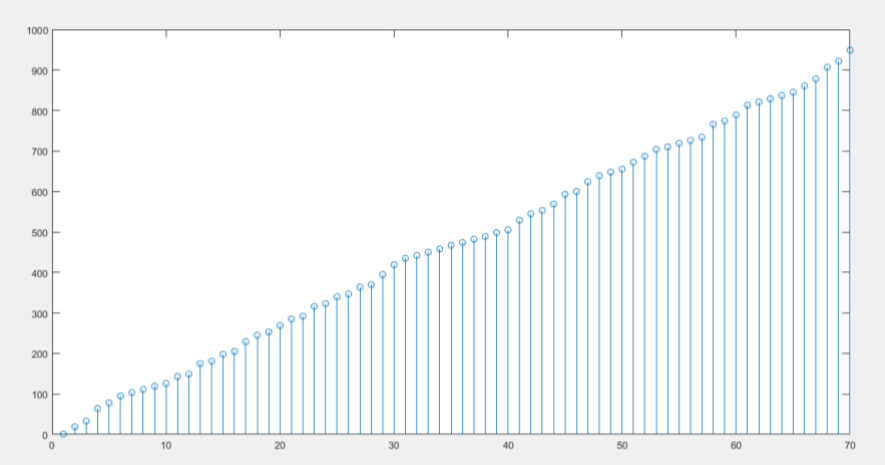

Figure 11: Stemplot of the locations where the edges occurs in the signal y[n]

(d)

```
MatLab code used in part (d):
>> delta_n = conv(ln,difh);
```

```
>> subplot(211), stem (ln)
>> subplot(212), stem(delta_n)
>> % most of the values of the deltan are 8,16,24,32
>> subplot(211), stem (ln)
>> subplot(212), stem(delta_n)
```

The convolution of l[n] with the first difference filter will give the width of the bars of the UPC in pixels. As shown in the stem plot in figure 12, the Δ[n] signal has pulses in the range of 8, 16, 24 and 32. As there are four different widths of the black and white bar in the barcode, the values of the pulses in the Δ[n] signal are around 8, 16, 24 and 32.

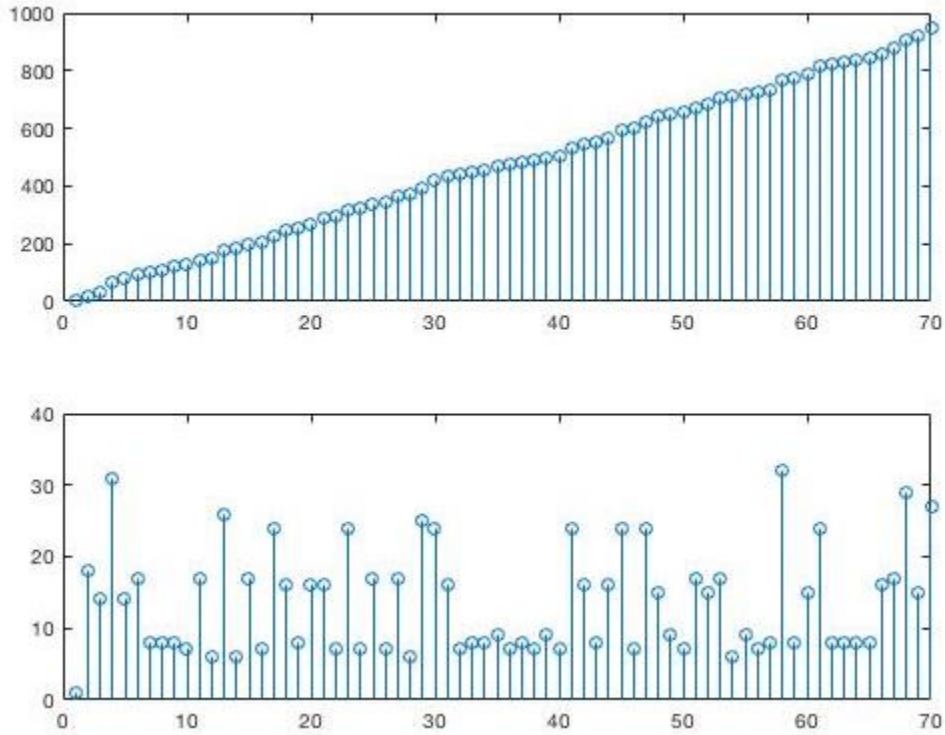

Figure 12: Stem plot of l[n] (top) and Δ[n] (bottom)

(e)

A 12 number UPC barcode has 95 thin bars. In 95 thin bars, each number includes 7 thin bars which makes 84 black and white thin bars. The barcode starts and ends with a group of three thin bars which makes six more bars. The middle of the bar has five thin black and white bars. Therefore, the barcode has 84+6+5 = 95 thin bars. If the width of the thin bar is $\theta_1$pixels, the total width of the UPC barcode with 12 numbers will be $95\theta_1$pixels. As there are 95 thin bars

where some bars has the width in multiple of the width of thin bar, the total width of the barcode with 95 thin bars of $\theta_1$pixels is $95\theta_1$.

(f)

```
MatLab code used in part (f):
>> sum = 0;
>> for n = 7:65
       sum = sum + delta_n(1,n);
end
>> divider = sum / 95; % divider = 7.8947368421052
```

The signal Δ[n] has the length of 1x71. The first six values of the 71 values in Δ[n] signal is not important and useless. The reason for the first six values being useless is the convolution process. In the convolution of the first difference filter, the convolution uses values of n = -1, 0. As the values of the input signal x[n] is not defined at n= -1 and 0, the first two convolution values are ignored. As the convolution process occurs two times, the first four values of the Δ[n] signals are ignored. In addition, the sixth and seventh value represent the width till the filter senses the first bar of the UPC barcode. Therefore, the first six values of the Δ[n] signal was ignored.

As shown in the matlab code above, the value of $\theta_1$was calculated by taking the average of the Δ[n] signal. The average was 7.89 approximately for θ_1.

(g)

```
MatLab code used in the part (g):
>>for n = 1:71
      delta_n(1,n) = delta_n(1,n)/divider;
>>for n=1:71
     if (delta_n(1,n) >0.5 && delta_n(1, n) <= 1.5)
               delta_n(1,n) = 1;
     end
     if (delta_n(1,n) >1.5 && delta_n(1, n) <= 2.5)
               delta_n(1,n) = 2;
     end
     if (delta_n(1,n) >2.5 && delta_n(1, n) <= 3.5)
               delta_n(1,n) = 3;
```

```
    end
    if (delta_n(1,n) >3.5 && deln(1, n) <= 4.5)
              delta_n(1,n) = 4;
    end
end
```

As shown in the matlab code block above, the values of Δ[n] signal was converted into the values 1,2,3 and 4 by dividing the values by the value of $\theta_1$and rounding them to the nearest integer values.

(h)

```
MatLab code used for part (h):
>> barcodearray = delta_n(1,7:65);
>> decodeUPC (barcodearray)
ssbeg =
     1     1     1
ans =
     8     8     2     7     8     0     4     5     0
1     6     5
```

Since the p-code function only receives an array of length 59 as an argument, the Δ[n] signal was converted to a barcodearray of length 59. Since the first six values of the Δ[n] signal were ignored, the barcodearray was made from the 59 values after the first six values of Δ[n] signal. The barcodearray was processed through the function decodeUPC which gives the barcode number.

(i) The UPC barcode shown in the figure 9 has the number 882780450165. The output of the p-code function decodeUPC also gave the same 882780450165 number for the barcode. Therefore, the process of decoding UPC was successful.

MatLab Code for Decoding 'HP10v3.png' Barcode:

```
>> bcode = imread('HP110v3.png');%Barcode Image %loading
>> xn = bcode(250, :);% row 250 was chosen randomly to
%process the barcode
>> difh = [1 -1];% difference filter
```

```
>> yn = conv (double(xn),double(difh));% Filtered xn
>> subplot(211), stem(xn)
>> subplot(212), stem(yn)
%The following for loop creates a threshold signal with
%zeros and ones where one represents an edge.
>> for m =1:949
      if (abs(yn(1,m)) < 50) % Threshold value: τ = 50
         dn(1,m) = 0;
      else
         dn(1,m) = 1;
      end
end
>> ln = find (dn);
>> stem (ln)
>> delta_n = conv(ln,difh);
>> subplot(211), stem (ln)
>> subplot(212), stem(delta_n)
>> % most of the values of the deltan are 8,16,24,32
>> sum = 0;
>> for n = 7:65
       sum = sum + delta_n(1,n);
end
>> divider = sum / 95; % divider(θ_l)= 7.8947368421052
>>for n = 1:71
      delta_n(1,n) = delta_n(1,n)/divider;
  end
>>for n=1:71
     if (delta_n(1,n) >0.5 && delta_n(1, n) <= 1.5)
               delta_n(1,n) = 1;
     end
     if (delta_n(1,n) >1.5 && delta_n(1, n) <= 2.5)
               delta_n(1,n) = 2;
     end
     if (delta_n(1,n) >2.5 && delta_n(1, n) <= 3.5)
               delta_n(1,n) = 3;
     end
     if (delta_n(1,n) >3.5 && delta_n(1, n) <= 4.5)
               delta_n(1,n) = 4;
     end
end
```

```
>> barcodearray = delta_n(1,7:65);
>> decodeUPC (barcodearray)
ssbeg =
     1     1     1
ans =
     8     8     2     7     8     0     4     5     0
1     6     5
```

(j) For this part of the experiment, we want apply the same method of decoding to a second image. Figure 13 shows the image to be decoded.

Figure 13: OFFv3 UPC Barcode

Notice that this image is skewed, this will add to the complexity of analysis, specifically the angle of the skew.

The UPC image shown in Figure 13 was loaded onto MATLAB and the 85th row was extracted and saved onto an 1x342 vectors labeled ‘xn’. The convolution of x[n] with first difference filter is saved into the 1x 343 vector labeled ‘yn’. The stem plot of x[n] and y[n] is shown below in figure 14.

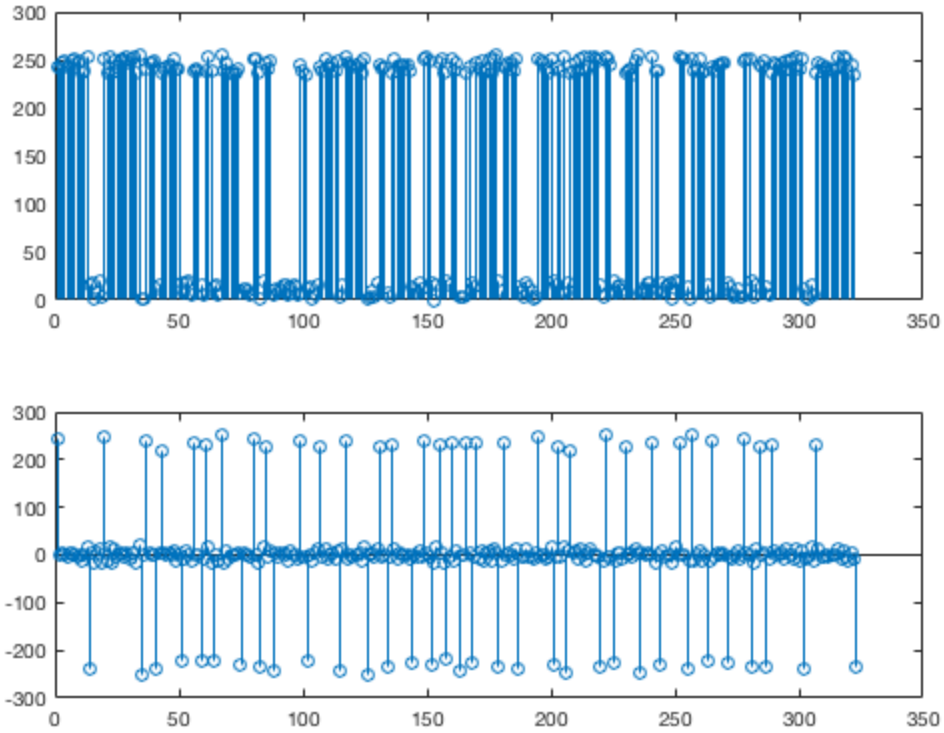

Figure 14: Stem Plot of 'xn' on top and the Filtered Step Plot 'yn' on the bottom

The threshold value of the signal was set to 50 as the magnitude of the pulses at the edge had the absolute value of around 250. Any pulse that has magnitude of less than 50 has to be ignored since the pulses with less than 50 magnitude did not represent the edge. The signal d[n] represents the threshold signal that contains the values of only zeros and ones where one represents an edge. The locations of the edge, locations of ones in the d[n] signal, occurrence was then saved into the signal l[n] using `find` function. The stem plot of the signal l[n] is showed in figure 15 below.

The convolution of l[n] with the first difference filter will give the width of the bars of the UPC in pixels. As shown in the stem plot in figure 16, the Δ[n] signal has pulses in the range of 3, 6, 9 and 12. As there are four different widths of the black and white bar in the barcode, the values of the pulses in the Δ[n] signal are around 3, 6, 9 and 12.

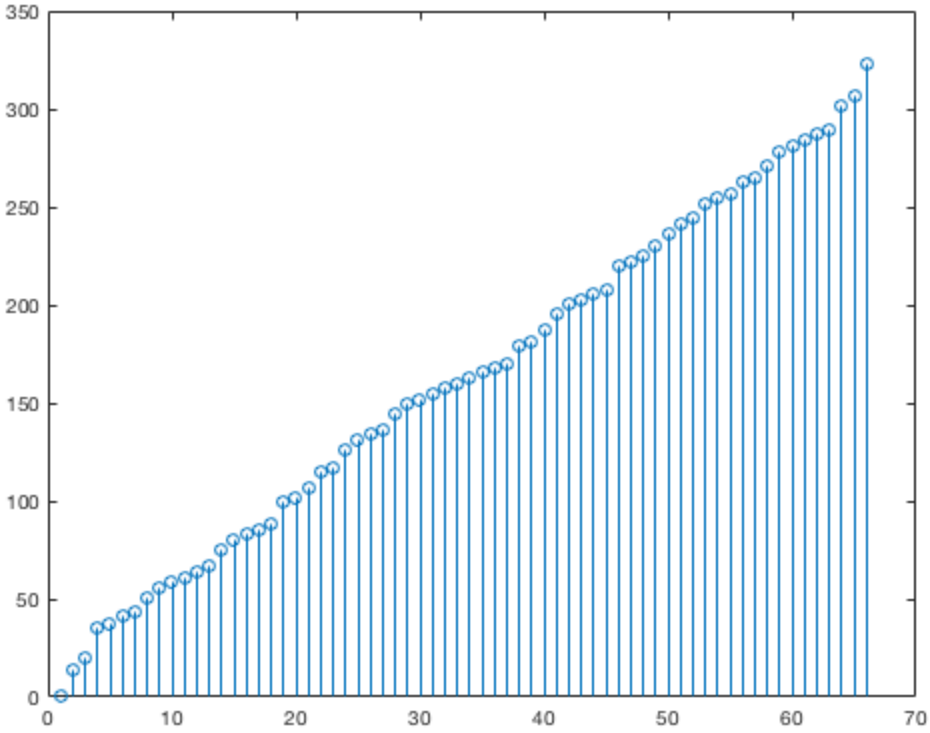

Figure 15: Stemplot of the locations where the edges occurs in the signal y[n]

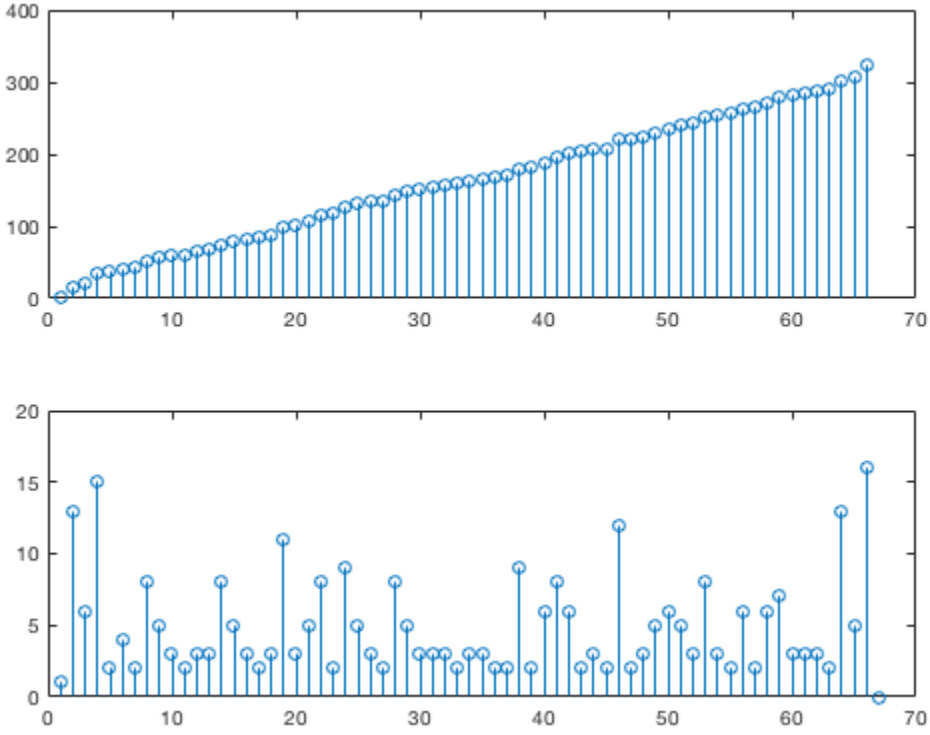

Figure 16: Stem plot of l[n] (top) and Δ[n] (bottom)

Edge Detection in Images: UPC Decoding

A 12 number UPC barcode has 95 thin bars. In 95 thin bars, each number includes 7 thin bars which makes 84 black and white thin bars. The barcode starts and ends with a group of three thin bars which makes six more bars. The middle of the bar has five thin black and white bars. Therefore, the barcode has 84+6+5 = 95 thin bars. If the width of the thin bar is $\theta_1$pixels, the total width of the UPC barcode with 12 numbers will be $95\theta_1$pixels. As there are 95 thin bars where some bars has the width in multiple of the width of thin bar, the total width of the barcode with 95 thin bars of $\theta_1$pixels is $95\theta_1$.

The signal Δ[n] has the length of 1x 67. The first four values of the 67 values in Δ[n] signal is not important and useless. The reason for the first four values being useless is the convolution process. In the convolution of the first difference filter, the convolution uses values of n = -1, 0. As the values of the input signal x[n] is not defined at n= -1 and 0, the first two convolution values are ignored. As the convolution process occurs two times, the first four values of the Δ[n] signals are ignored.

As shown in the matlab code below, the value of $\theta_1$was calculated by taking the average of the Δ[n] signal. The average was 2.67 approximately for θ_1. The values of Δ[n] signal was converted into the values 1,2,3 and 4 by dividing the values by the value of $\theta_1$and rounding them to the nearest integer values.

Since the p-code function only receives an array of length 59 as an argument, the Δ[n] signal was converted to a barcodearray of length 59. Since the first six values of the Δ[n] signal were ignored, the barcodearray was made from the 59 values after the first six values of Δ[n] signal. The barcodearray was processed through the function decodeUPC which gives the barcode number.

MatLab Code for Decoding 'OFFv3.png' Barcode:

```
>>bcode = imread('OFFv3.png');%Barcode Image loading
>>xn = bcode(85, :);% row 250 was chosen randomly to process
%the barcode
>>difh = [1 -1];% difference filter
>>yn = conv (double(xn),double(difh));% Filtered xn
>>subplot(211), stem(xn)
>>subplot(212), stem(yn)
%The following for loop creates a threshold signal with zeros
% and ones where one represents an edge.
>>for m =1:323
      if (abs(yn(1,m)) < 50) % Threshold value: =50
```

```
         dn(1,m) = 0;
      else
         dn(1,m) = 1;
      end
end
>>ln = find (dn);
>>stem (ln)
>>delta_n = conv(ln,difh);
>>subplot(211), stem (ln)
>>subplot(212), stem(delta_n)
% most of the values of the deltan are 3,6,9,12
>>subplot(211), stem (ln)
>>subplot(212), stem(delta_n)
>>sum = 0;
>>for n = 5:63
       sum = sum + delta_n(1,n);
end
>>divider = sum / 95; % divider(θ_l) = 2.67368421052632
>>for n = 1:67
      delta_n(1,n) = delta_n(1,n)/divider;
end
>>for n=1:67
    if (delta_n(1,n) >0.5 && delta_n(1, n) <= 1.5)
            delta_n(1,n) = 1;
    end
    if (delta_n(1,n) >1.5 && delta_n(1, n) <= 2.5)
            delta_n(1,n) = 2;
    end
    if (delta_n(1,n) >2.5 && delta_n(1, n) <= 3.5)
            delta_n(1,n) = 3;
    end
    if (delta_n(1,n) >3.5 && delta_n(1, n) <= 5)
            delta_n(1,n) = 4;
    end
end
>>barcodearray = delta_n(1,5:63);
>>decodeUPC (barcodearray)
ssbeg =
     1     1     1
ans =
  Columns 1 through 10
     0     4     6     5     0     0     7     0     3     1
  Columns 11 through 12
     9     5
```

As shown at the end of the matlab code above, the decoded value of the UPC from the skewed barcode shown in figure 13 is 046500703195.