DSP - Laboratory 1

Daniele Orsuti
daniele.orsuti@phd.unipd.it

October 27, 2021

Abstract

First laboratory experience [1] for the course Digital Signal Processing (DSP) a.y. 21/22. Topics: filters, their applications, and fundamentals of image processing.

1 Universal Product Code (UPC) decoding - Edge detection in images [1]

A 12-digit bar code consists of alternating black and white bars; the white bars appear to be spaces. The UPC (Universal Product Code) *uses widths of the bars to encode numbers.* There are four widths that are integer multiples of the thinnest black bar, or thinnest white space. Thus, we define a 3-wide black bar as three times as wide as the thinnest black bar; likewise, for 2-wide and 4-wide bars—whether black or white. Look at any bar code (Fig. (1)), and you should be able to identify the four widths. Each number from 0 to 9 is encoded with a quadruplet. Here is the encoding of the digits 0–9:

$$
\begin{aligned}
0 &= 3-2-1-1; \quad 5 = 1-2-3-1 \\
1 &= 2-2-2-1; \quad 6 = 1-1-1-4 \\
2 &= 2-1-2-2; \quad 7 = 1-3-1-2 \\
3 &= 1-4-1-1; \quad 8 = 1-2-1-3 \\
4 &= 1-1-3-2; \quad 9 = 3-1-1-2
\end{aligned}
$$

Figure 1: UPC code.

For example, the code for the number "5" is 1-2-3-1, meaning it could be a one-unit wide white space, followed by a 2-wide black bar, followed by a 3-wide white space, and finally a 1-wide black bar (or inverted: 1-wide black, 2-wide white, 3-wide black, and 1-wide white). The UPC (Universal Product Code) consists of twelve digits delimited on each end by 1-1-1 (blackwhite-black), and separated in the middle (between the sixth and seventh digit) by white-black-white-blackwhite (1-1-1-1-1). Thus the entire UPC must have 30 black bars and 29 white bars for a total of 59. Furthermore, note that the encoding for each digit always adds up to seven so the total width of the bar code is always the same.

[1] All information and material presented here is intended to be used for educational or informational purposes only.

In terms of the unit width where the thinnest bars have width one, it should be easy to determine that the total width of the UPC bar code is 95 units [2].

Follow the steps below to develop the processing needed to decode a typical bar code from a scanned image. The following description refers to figure (2).

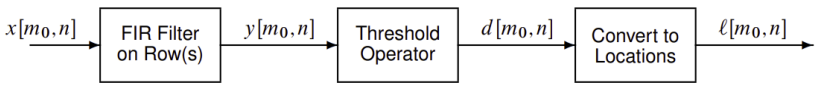


Figure 2: Using an FIR system plus a threshold operator to detect and locate edges, i.e., transitions. m_0: index of the selected row.

A decoder for the final step is provided as a MATLAB p-code file (called *decodeUPC.p*). The data files and the decoder for the lab are available in the folder.

- Read the image *upca5.gif* into MATLAB with the imread function. Extract one row (in the middle) to define a 1-D signal $x[n]$.

- **FIR Filter on Row**
  Filter the signal $x[n]$ with a first-difference FIR filter (output: $y[n] = x[n] - x[n-1]$ (impulse response written as $[1, -1]$ in MATLAB)); call the output $y[n]$. Use *conv* with option *'valid'* to perform the filtering operation and remove the part of the convolution computed on the zero-padded edges. Make a stem plot of the input and output signals, in the same figure window. *Note:* we will mark only one side of the transition as true when you threshold the output of the first-difference filter. Is it located before or after the transition?

- Create a sparse detected signal $d[n]$ by comparing the magnitude $|y[n]|$ to a threshold. Then convert the sparse signal $d[n]$ into a location signal $l[n]$ by using the find function to extract locations. Make a *stem* plot of the location signal, $l[n]$.

  Threshold operator:

  $$d(n) = \begin{cases} \text{Edge False}, & \text{if } |y[n]| < \tau \\ \text{Edge True}, & \text{if } |y[n]| \geq \tau \end{cases}$$

  Determine an appropriate value of the threshold τ to get the edges. In MATLAB use the *abs* function along with a logical operator (such as $>$ or $<$) to define this thresholding system that gives a "TRUE" binary output for the edges, with $y[n]$ as the input to this thresholding system.

  Convert to locations: Use MATLAB's *find* function to produce a shorter signal that contains the edge locations; *Note:* The length of the location signal must be greater than or equal to 60, if the rest of the processing is going to succeed.

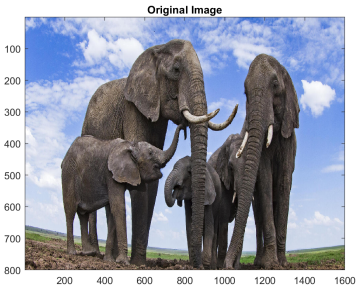
- Apply a first-difference filter to the location signal; call the output $\Delta[n]$. These differences should be the widths of the bars. Make a stem plot of $\Delta[n]$, and put this plot and the previous one of $l[n]$ in the same figure window by using subplot. Observe how the plot of $\Delta[n]$ conveys the idea that there are (approximately) four different widths in the bar code.

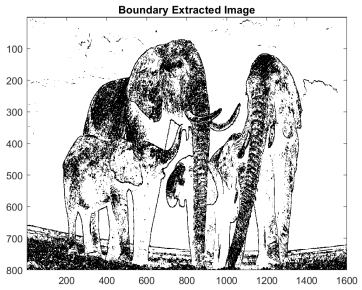
- In order to successfully decode the UPC code, it is necessary to estimate the the basic width (w_b), in pixels, of the thinnest bar and use that to normalize $\Delta[n]$. From the introduction it is known that the total width of a valid 12-digit bar code is equal to $95w_b = W$, where W is the total width in number of pixels of the code. Therefore the above equation can be inverted to find w_b. Find the value of W by using $\Delta[n]$ (hint: look at how $\Delta[n]$ has been defined) or $l[n]$ and then compute w_b. Check if the value of w_b is correct by looking directly at the width (in pixel) of the thinnest bar from the imported image.

[2]Solution: 12 digits $\times$ 7 thin bars per digits + 3 thin bars at the beginning + 3 thin bars at the end + 5 thin bars in the middle. For more information visit: http://electronics.howstuffworks.com/gadgets/high-tech-gadgets/upc3.htm.

- Using your estimate of w_b from the previous part, convert the values of Δ_n into relative sizes by dividing by w_b and rounding. The result should be integers that are equal to 1, 2, 3 or 4, assuming you are analyzing a valid bar code.
- Now we are ready to perform the decoding to digits. A p-code function *decodeUPC* is provided for that purpose. It takes one input vector that has to be a length-59 vector of integers, i.e., the output of the previous part. The function *decodeUPC* has one output which should be a vector of twelve single-digit numbers, if the decoder does not detect an error. When there is an error the output may be partially correct, but the incorrect decodes will be indicated with a -1. *Note:* The signal entering the decoder should have 3 ones on its head and 3 ones on its tail, if the previous steps have been done correctly.
- The 12 digits which are expected are give in the loaded image. Test all the images that are available. Remember to convert the image from RGB to black and white if necessary, and to check if the value of τ needs to be updated.

2 Edge detection in images using local variance [2]





Local statistics can determine the gradient of the image. The local variance can be used to generate an edge map. In this section we first compute the local variance of an image, then we find the edges by exploiting the local variance. Local variance means that instead of finding variance for the whole matrix which represent the image, variance is computed based on a small sliding window. Figure 3 shows the steps which allow to compute the local variance and that we are going to implement in MATLAB. Before following the steps in figure 3, load the image *eleph2.jpg*, convert it to gray scale (*rgb2gray(·)*), then to double (*double(·)*), and finally plot the figure using the following line

```
figure,imagesc(I);colormap(gray);title('Original Image B&W');
```

Once the steps in figure 3 have been completed, proceed with edge detection:

- Find the mean of the local variance matrix and save the results to use it as a threshold for edge detection (The threshold value can also be set by the user. For instance, set the threshold value to 400.)
- Apply thresholding to the local variance matrix with the threshold found on previous step.
- Show the results using *imagesc and colormap as above.*

What does it remind you the procedure you have implemented to compute the local variance? Can you guess an alternative implementation by using default MATLAB functions?

1. Consider a matrix X= $\begin{bmatrix} 5 & 3 & 7 & 2 \\ 11 & 3 & 22 & 8 \\ 9 & 2 & 8 & 22 \\ 6 & 5 & 4 & 4 \\ 7 & 3 & 1 & 4 \end{bmatrix}$

2. Define a window of size 3x3

3. Pad with zeros on all sides $\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 5 & 3 & 7 & 2 & 0 \\ 0 & 11 & 3 & 22 & 8 & 0 \\ 0 & 9 & 2 & 8 & 22 & 0 \\ 0 & 6 & 5 & 4 & 4 & 0 \\ 0 & 7 & 3 & 1 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$

4. Find the local mean by sliding 3x3 window on the matrix

$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 5 & 3 & 7 & 2 & 0 \\ 0 & 11 & 3 & 22 & 8 & 0 \\ 0 & 9 & 2 & 8 & 22 & 0 \\ 0 & 6 & 5 & 4 & 4 & 0 \\ 0 & 7 & 3 & 1 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$ =(0+0+0+0+5+3+3+11)/9=2.444

Replace the middle value with 2.444

Slide the window to the next position and find the mean.
Repeat this process for the whole matrix.

$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 5 & 3 & 7 & 2 & 0 \\ 0 & 11 & 3 & 22 & 8 & 0 \\ 0 & 9 & 2 & 8 & 22 & 0 \\ 0 & 6 & 5 & 4 & 4 & 0 \\ 0 & 7 & 3 & 1 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$ =5.6667

5. The local mean of the matrix X =

$\begin{bmatrix} 2.4444 & 5.6667 & 5.0000 & 4.3333 \\ 3.6667 & 7.7778 & 8.5556 & 7.6667 \\ 4.000 & 7.7778 & 8.6667 & 7.5556 \\ 3.5556 & 5.000 & 5.8889 & 4.7778 \\ 2.3333 & 2.8889 & 2.3333 & 1.4444 \end{bmatrix}$

6. Find the local mean for X^2 = $\begin{bmatrix} 18.2222 & 77.4444 & 68.7778 & 66.7778 \\ 27.6667 & 94.000 & 130.1111 & 127.6667 \\ 30.6667 & 93.3333 & 129.5556 & 125.3333 \\ 22.6667 & 31.6667 & 70.5556 & 66.3333 \\ 13.2222 & 15.1111 & 9.2222 & 5.4444 \end{bmatrix}$

7. Local Variance = Local mean of X^2 – (Local mean of X)^2

$\begin{bmatrix} 18.2222 & 77.4444 & 68.7778 & 66.7778 \\ 27.6667 & 94.000 & 130.1111 & 127.6667 \\ 30.6667 & 93.3333 & 129.5556 & 125.3333 \\ 22.6667 & 31.6667 & 70.5556 & 66.3333 \\ 13.2222 & 15.1111 & 9.2222 & 5.4444 \end{bmatrix}$ –

$\begin{bmatrix} 2.4444 & 5.6667 & 5.0000 & 4.3333 \\ 3.6667 & 7.7778 & 8.5556 & 7.6667 \\ 4.0000 & 7.7778 & 8.6667 & 7.5556 \\ 3.5556 & 5.000 & 5.8889 & 4.7778 \\ 2.3333 & 2.8889 & 2.3333 & 1.4444 \end{bmatrix}^2$ =

$\begin{bmatrix} 12.24 & 45.33 & 43.77 & 48.00 \\ 14.22 & 33.50 & 56.91 & 68.88 \\ 14.66 & 32.83 & 54.44 & 68.24 \\ 10.02 & 6.66 & 35.87 & 43.50 \\ 7.77 & 6.76 & 3.77 & 3.35 \end{bmatrix}$

Figure 3: Steps to follow.

3 Auto-Correlation and simple Echo Cancellation [3]

Now you will use the correlation of two sequences to solve a simple echo cancellation problem. The correlation of ***real*** sequences $y[n]$ and $z[n]$ is another sequence, say $R_{yz}[n]$, that is defined by

$$R_{yz}(n) = y[n] * z[-n]$$

That is, it is computed by evaluating the convolution of $y[n]$ with the time-reversed version of $z[n]$. The correlation serves as a useful measure of similarity between two sequences (and is actually used in pattern recognition applications).

When both sequences $y[n]$ and $z[n]$ are identical, we refer to the resulting correlation sequence as the auto-correlation sequence. Therefore, the auto-correlation sequence of $y[n]$ is simply the convolution of $y[n]$ with $y[-n]$, and is denoted by

$$R_y[n] = y[n] * y[-n]$$

Now consider a signal $y[n]$ that is obtained as the sum of a sequence $x[n]$ and its delayed and scaled version, $ax[n-N]$,

$$y[n] = x[n] + a \cdot x[n-N]$$

where a is the amplitude of an echo $x[n-N]$ and N is the delay in samples. Given measurements of $y[n]$, we are interested in estimating the delay N of the echo. In the sequel we assume that the original sequence $x[n]$ starts at time $n = 0$. That is, $x[n]$ is zero for $n < 0$.

Due to the form of $y[n]$, its autocorrelation function can be shown to have peak values at the time instants $n = 0$, $n = N$, and $n = -N$ (you will verify this during the exercise).

Therefore, from a plot of the auto-correlation function we can determine the delay N. We assume a is known (methods for estimating a are beyond the scope of this exercise). Now given a and N, their values can be used to remove the echo and recover the original signal. This can be achieved in several ways, here a one method is presented.

3.1 Method

First recall that we are assuming that the original sequence $x[n]$ starts at time n=0. That is, $x[n]$ is zero for $n < 0$. This shows that the first N samples of $x[n]$ and $y[n]$ coincide,

$$x(n) = y(n) \quad \text{for} \quad n < N.$$

We still need to recover the values of $x[n]$ for time instants larger than or equal to N. For this purpose, we simply note that

$$y[N] = x[N] + a \cdot x[0]\,,$$

so that $x[N]$ can be recovered from

$$x[N] = y[N] - a \cdot x[0]\,.$$

Likewise, for values of n larger than N we employ successively the following relation,

$$x[n] = y[n] - a \cdot x[n-N]\,.$$

3.2 Tasks

- Load the audio signal *y_echo.mat* which is sampled at $F_s = 44100$ Hz. Play the audio with the command *sound* (please, remember to specify F_s).
- Compute the autocorrelation of the audio signal by using the MATLAB function *xcorr*, and plot the results obtained.
- From the plot obtained in the previous point, find the value N defined above (Sec. (3)).
- Apply the technique presented in methods (Sec. 3.1) to cancel the echo from the audio signal (for this task consider $a = 0.75$). *Hint:* follow Sec. 3.1 step by step.
- Plot, using *subplot*, the original and the filtered signal, then play them and check the result.

  $$x[n] = y[n] - a \cdot x[n-N]\,.$$

  Is it an FIR filter?

References

[1] J. H. McClellan, R. W. Schafer, and M. A. Yoder. *DSP First (2nd Edition)*. Prentice-Hall, Inc., USA, 2007.

[2] not defined. imageprocessinglab. Report, not defined, 2021.

[3] A. H. Sayed. Dsp lab. Report, UCLA Electrical Engineering Department, 2000.