Development of a Barcode Reader System

Firas Abdullah Thweny Al-Saedi, Ali Al-Bayaty

Computer Engineering Department, Al-Nahrain University, Baghdad, Iraq

Abstract *— In this paper, a software package was developed to decode a defined set of barcode standards. This package captures the barcode image by using a commercial USB (Universal Serial Bus) Webcam device, color-scale or grayscale of capturing modes. Several digital image processing filters were used to find the barcode area (inside the captured image) and to enhance the founded barcode area. These filters were based on the known techniques of the digital image processing, like: thresholding, differencing, averaging and histogram-equalization. Finally, the enhanced and founded barcode area is decoded on the human-readable digits and displayed to the PC monitor. Because of the various barcode types (or symbologies), this package is capable of decoding three (1D (One dimensional) numeric character set, continuous, multiple-width, fixed-length, self-checking) barcode symbologies: EAN-13 (European Article Numbering-with 13 digits), UPC-A (Universal Product Code–Class A), and ISBN (International Standard Book Number). Microsoft DirectShow 8.0 was used as the software tool for the USB Webcam device driver interface and Microsoft Visual C++ 6.0 GUI (Graphical User Interface) as the software language for the digital image processing filters and for the decoding algorithms of the defined three barcode symbologies. Many tests on package were done, and the obtained results were satisfactory in terms of accuracy and speed.*

Keywords *— Barcode reader, EAN-13, 1D barcode, ISBN reader, UPC-A, Image processing.*

I. INTRODUCTION

The development of this software package depends mainly on the capturing and decoding operations that applied on the barcode image.

The software was implemented and achieved by using the algorithms of: (a) Capturing image routines [1][2] (capturing the barcode image via a USB Webcam) and (b) Decoding procedures of a defined set of barcode symbologies (e.g. EAN-13, UPC-A and ISBN) [3][4]. Digital image processing techniques were used to enhance the captured barcode image and to give it a custom features to facilitate the decoding procedures [5].

The paper can be categorized into three parts: Part I consists of the USB Webcam device driver interface that is used to capture an image (a barcode image), Part II deals with the techniques or filters of the digital image processing that is applied to the captured image (e.g. grayscale, threshold, histogram-equalization and so on) and Part III which consists to the decoding procedures of the barcode and to display the resultant decoded barcode number. The functional block diagram shown in Figure 1, illustrates the steps of the system implementation with the corresponding parts.

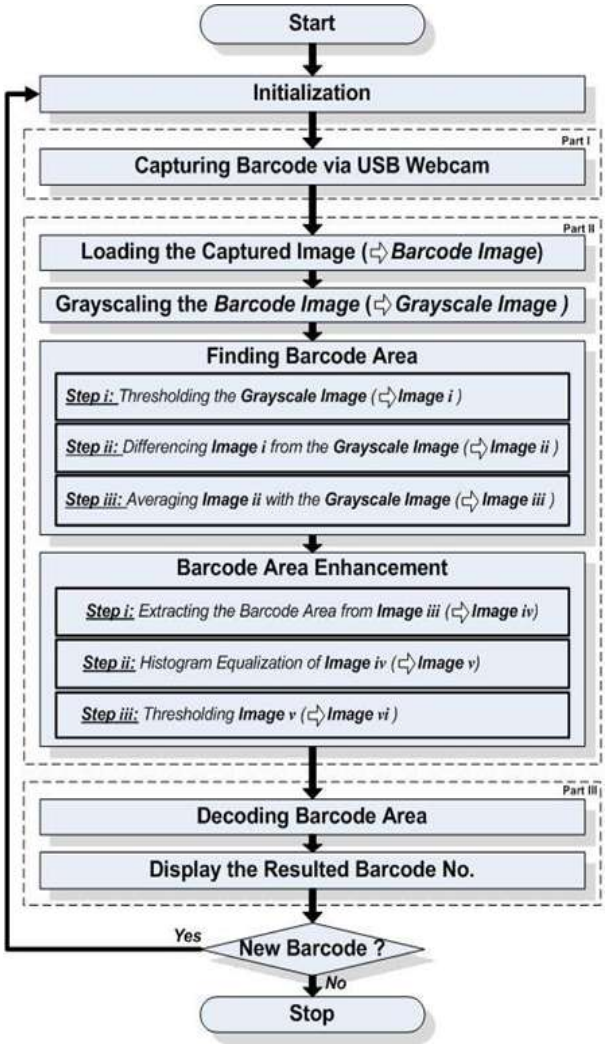

Fig. 1 The functional block diagram

II. SYSTEM LAYOUT

The overall system layout is illustrated as shown in Figure 2, with the aid of the functional block diagram of Figure 1.

This section and the following sections describe the whole operations of the software package; these descriptions will depend upon the sequences of the functional block diagram of Figure 1. This functional block diagram is divided into three parts, each part accomplishes a specific task.

The output of each part will be handled as input to the next part, with the exception of Part I which handles a variable pre-defined inputs.

The operation of each part is divided into steps, for the simplicity and easily illustration of that operation.

An operation or a step is explained with the aid of the histogram illustrations and pseudocodes (mixture of English and programming language, such as C++ or Java that used to describe a step line-by-line for high explanation for the actual operations).

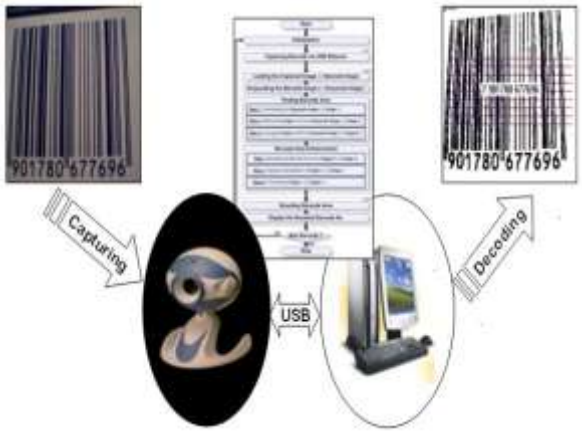

Fig. 2 The proposed system layout

A. Initialization

The first and the main operation of the software is the initialization process. It is used to initialize the package for the first run, to prepare the pre-defined inputs to Part I and to manage the restarting routine for the software (when the end-user wants to restart the software for another barcode processing). So that, when the software starts running for the first time, the initialization procedures are not as the procedures when the software restarting, as shown in Figure 1.

The initialization process is divided into different processes, which are:

- Program Initialization: holds the GUI (Graphical User Interface), image viewers, histograms viewers and the information that related for each step of the process.
- DirectX Initialization: contains codes and filters for Webcam finding, previewing and capturing.
- Variable Declaration: declares variables and constants in the software.
- Variable Assignment: initializes the previous variables and reinitializes them if the software is restarted.

The complete steps in this operation are illustrated in the following pseudo code:

```
//Initialization Procedure Pseudocode:
BEGIN
//Program Initialization...
//For each step there are image viewers.
INITIALIZE image-viewers
//For each step there are histograms.
INITIALIZE histograms
//A text info for each step.
INITIALIZE info
//DirectX Initialization...
INITIALIZE directx-object
//Initializing the filters of: finding Webcam
//device, previewing and capturing.
INITIALIZE directx-filters
//Variables Declaration...
//Pixels for digital image processing.
PIXEL pixel, pixel2
//For every step there are image files.
IMAGE img-capture, im-gray, img-i, img-ii,
      img-iii, img-iv, img-v, img-vi
//Declare image's width and height for each
image file.
VARAIBLE img-width, img-height
//Barcode specifications in an image.
VARAIBLE barcode-width, barcode-
height,barcode-size, barcode-number
//Rectangle's dimensions.
VARAIBLE rectYo, rectXo, rectYn, rectXn
//Iterator counters.
VARAIBLE X, Y
//Temporary storages.
VARAIBLE index, swap, scan-line[20]
//Each histogram has a counter of color
//appearance in the image.
VARAIBLE histogram-counter
//Number of pixels per color.
VARAIBLE Nk[16]
//Probability Density Function.
VARAIBLE Pr[16]
//Transformation Function of the Histogram
//Equalization.
VARAIBLE S[16]
//For checking the states either true or
false.
BOOL done
//Creating the Grayscale Look-Up table...
CONSTANT grayscale-table[16]
FOR X=1 TO X=15   STEP1
grayscale-table[X]=16*X , grayscale-
table[16]=255
//Variables Assignment...
//From this position, the program reinitiates
when it restarted.
restart:
```

```
//Assuming all the states are false
done=FALSE
//Zeros all the arrays.
FOR X=1 TO X=16 STEP1
Nk[X]=Pr[X]=S[X]=0
END
```

B. Capturing barcode via USB webcam

The capturing operation is done by using Microsoft DirectShow, and the procedures of capturing the image are:

- Finding the connected USB Webcam(s).
- Displaying the found USB Webcam device name.
- Starting the previewing until the beginning of the capturing procedure (the capturing process may be in the Standard Mode or in the Mirror and Flip Mode).
- Transfer the captured image to Part II.

The following pseudocode illustrates this step:

```
//Capturing Barcode via USB Webcam
Pseudocode:
BEGIN
//Activate finding Webcam device filter.
done = LOAD find-webcam
//When there is a Webcam device.
IF done = TRUE THEN {
  //Activate the name specification filter.
  LOAD device-name
  //Activate the capturing filter.
  LOAD capture
  //Activate the previewing filter.
  LOAD preview
  //Displaying Webcam device name.
  ATTACH device-name (find-webcam)
  //Start previewing.
  RUN preview (find-webcam)
  //Here, the end-user is ready to press the
  //capture button...
  done = capture-button-click
  IF done = TRUE  THEN {
    //Pause previewing.
    PAUSE preview (find-webcam)
    //Start capturing and attaching the
    //captured image to a file.
    img-capture = RUN capture (find-webcam)
    //Transfer the captured image "file" to
    //Part II.
    TRANSFER img-capture
     //Stop previewing.
    STOP preview (find-webcam)
    //Stop capturing.
    STOP capture (find-webcam)
    //Deactivate previewing filter.
    UNLOAD preview
    //Deactivate capturing filter.
    UNLOAD capture
    //Deactivate the name specification
filter.
    UNLOAD device-name
  }
}
//Deactivate finding webcam device filter.
ELSE (UNLOAD find-webcam)
  MESSAGE "There is no Webcam device!!!"
//Deactivate finding Webcam device filter.
UNLOAD find-webcam
END
```

C. Loading barcode image

The captured image has been transferred and loaded to Part II, (the image viewer), which will be processed by the digital image processing filters later.

The following pseudocode illustrates this step:

```
//Loading Barcode Image Pseudocode:
BEGIN
//Setting info for this operation.
SET info
//Clearing the image viewer.
SHOW image-viewer (EMPTY)
//Resetting the histograms.
SHOW histogram (EMPTY)
//Loading the captured image.
SHOW image-viewer (img-capture)
//Calculating image's width.
SET img-width (img-capture)
//Calculating image's height.
SET img-height (img-capture)
//Colors counter.
SET histogram-counters (img-width, img-
height)
//Calculating the histograms.
SHOW histogram (histogram-counters)
END
```

D. Grayscaling the barcode image

The first used digital image processing filter is the grayscale filter, this filter converts the captured color image of the RGB "24-bit" colors to a grayscale image of a 16 grayscale "4-bit" color quality as shown in Equation 1.

COLOR = (Red , Green , Blue)
(1)

The algorithm of the grayscale filter depends on the lookup table which contains 16 grayscale colors, so that, the software picks a pixel from a captured color image, compares it with the table and assigns the selective grayscale color to the picked pixel, and so on.

The following pseudocode illustrates this step:

```
//Grayscaling the Barcode Image Pseudocode:
BEGIN
//Setting info for this operation.
SET info
//Loop for the image's height.
FOR X=1 TO X=img-height STEP 1
{
  //Loop for the image's width.
  FOR Y=1 TO Y=img-width STEP 1
  { //Take pixel from captured image.
    pixel=img-capture.GETPIXEL(Y,X)
    //Find and procedure on the LOOK-UP
table.
    pixel2 = FIND pixel ON grayscale-
table[16]
    REPLACE pixel BY pixel2
    //Put pixel in the gray image.
```

```
    img-gray.SETPIXEL(pixel,Y,X)
  }
}
//Loading the gray image.
SHOW image-viewer (img-gray)
//Calculating image's width.
SET img-width (img-capture)
//Calculating image's height.
SET img-height (img-capture)
//Colors counter.
SET histogram-counters (img-width, img-
height)
//Calculating the histogram.
SHOW histogram (histogram-counters)
END
```

E. Finding barcode area

Three steps are included inside this operation, which are:

i. Threshold the output image of subsection (*D*) at 128 grayscale color (many grayscale levels were taken and by experiment, it is found that the suitable threshold level is at 128).

The following pseudocode illustrates this step:

```
//Finding Barcode Area Step i Pseudocode:
BEGIN
//Setting info for this step.
SET info
//Loop for the image's height.
FOR X=1 TO X=img-height STEP 1
{ //Loop for the image's width.
  FOR Y=1 TO Y=img-width STEP 1
  { //Take pixel from gray image.
       pixel=img-gray.GETPIXEL(Y,X)
    //Threshold pixel at 128.
       IF pixel >= 128  THEN
       //Converts to white color.
         Pixel = 255
       //Converts to black color.
       ELSE pixel = 0
    //Put pixel in image of step i.
    img-i.SETPIXEL(pixel,Y,X)
  }
}
//Loading the image of step i.
SHOW image-viewer (img-i)
//Colors counter.
SET histogram-counters (img-width, img-
height)
//Calculating the histogram.
SHOW histogram (histogram-counters)
END
```

ii. Difference the output image of Step ***i*** from the grayscale image of subsection (*D*). In this step every pixel from previous step's image is subtracted from the corresponding pixel of the grayscale image and absoluted the result as in Equation 2.

Pixel (Step ***ii*** Image) = | [Pixel(Grayscale Image) – Pixel(Step ***i*** Image)] |

(2)

The following pseudocode describes this step:

```
//Finding Barcode Area Step ii Pseudocode:
BEGIN
//Setting info for this step.
SET info
//Loop for the image's height.
FOR X=1 TO X=img-height STEP 1
{
    //Loop for the image's width.
    FOR Y=1 TO Y=img-width STEP 1
    {
      //Take pixel from gray image.
      pixel=img-gray.GETPIXEL(Y,X)
      //Take pixel from step i image.
      pixel2=img-i.GETPIXEL(Y,X)
      //Applying Eq. 2
      pixel = ABS (pixel - pixel2)
      //Put pixel in image of step ii.
      img-ii.SETPIXEL(pixel,Y,X)
    }
}

//Loading the image of step ii.
SHOW image-viewer (img-ii)
//Colors counter.
SET histogram-counters (img-width, img-
height)
//Calculating the histogram.
SHOW histogram (histogram-counters)
END
```

iii. Average the output image of Step ***ii*** with the grayscale image of subsection (*D*). In this step every pixel from previous step's image is averaged with the corresponding pixel of the grayscale image. This will create a homogenous area around the barcode (the Quiet Zone) as show in Equation 3. It will produce a "barcode area" at 127 grayscale level.

Pixel (Step ***iii*** Image) = [Pixel(Grayscale Image) + Pixel(Step ***ii*** Image)] / 2

(3)

The following pseudocode describes this step:

```
//Finding Barcode Area Step iii Pseudocode:
BEGIN
//Setting info for this step.
SET info
//Loop for the image's height.
FOR X=1 TO X=img-height STEP 1
{
  //Loop for the image's width.
  FOR Y=1 TO Y=img-width STEP 1
  {
    //Take pixel from gray image.
    pixel=img-gray.GETPIXEL(Y,X)
    //Take pixel from step ii image.
    pixel2=img-ii.GETPIXEL(Y,X)
    //Applying Eq. 3
    pixel = (pixel + pixel2)/2
    //Put pixel in image of step iii.
```

```
     img-iii.SETPIXEL(pixel,Y,X)
  }
}
//Loading the image of step iii.
SHOW image-viewer (img-iii)
//Colors counter.
SET histogram-counters (img-width, img-
height)
//Calculating the histogram.
SHOW histogram (histogram-counters)
//Now, searching for the homogenous area at
127 //level...
//Loop for the image's height.
FOR X=1 TO X=img-height STEP 1
{//Loop for the image's width.
    FOR Y=1 TO Y=img-width STEP 1
    { //Take pixel from step ii image.
      pixel=img-iii.GETPIXEL(Y,X)
      //Checking the 127 level.
      IF pixel = 127  THEN
        {//Draw the barcode area with the
    yellow
        //color.
        Pixel = YELLOW
           //Put pixel in step iii image.
        img-iii.SETPIXEL(pixel,Y,X)
        }
    }
}
//Identify from the left-top direction of
img-iii
FOR X=1 TO X=img-height STEP 1
{
  FOR Y=1 TO Y=img-width STEP 1
  {
    //Take pixel from step ii image.
    pixel=img-iii.GETPIXEL(Y,X)
    //Checking the 127 level.
    IF pixel = YELLOW  THEN
        {
        //Draw the barcode area with get
first
         //point of a rectangle.
        rectYo=Y , rectXo=X
        //Stop and exit this iteration.
        GOTO second
    }
  }
}
second:
//Identify from the right-botton direction of
//the img-iii.
FOR X= img-height TO X=1 STEP -1
{
  FOR Y= img-width TO Y=1 STEP -1
  {
    //Take pixel from step ii image.
    pixel=img-iii.GETPIXEL(Y,X)

    //Checking the 127 level.
    IF pixel = YELLOW  THEN
        {
        //Draw the barcode area with get
         //second point of rectangle.
        rectYn=Y , rectXn=X
        //Stop and exit this iteration.
        GOTO finish
    }
  }
}
finish:
//Now, having the two points of the desired
//rectangle...
DRAW RECTANGLE (rectYo, rectXo, rectYn,
rectXn,
                RED)
//Get the barcode area width.
barcode-width = rectYn – rectYo
//Get the barcode area height.
barcode-height = rectXn – rectXo

//Note that the histogram is taken before
changing the barcode area to the yellow
color. So that, the histogram of this step is
not affect and not count the variation of
//this color.

END
```

F. Barcode area enhancement

Three steps are included inside this operation, which are:

i. Extract the Barcode Area from the final output image of subsection (*E*) (Step ***iii***).

The following pseudocode illustrates this step:

```
//Barcode Area Enhancement Step i Pseudocode:
BEGIN
//Setting info for this step.
SET info

//Loop for the barcode's height.
FOR X=1 TO X=barcode-height STEP 1
{
  //Loop for the barcode's width.
  FOR Y=1 TO Y=barcode-width STEP 1
  {
    //Take pixel from img-iii.
    pixel=img-iii.GETPIXEL(Y,X)

    //Check for the yellow color.
    IF pixel = YELLOW  THEN
        //Converts to 127 color.
        Pixel = 127
       //Put pixel in this step image.
    img-iv.SETPIXEL(pixel,Y,X)
  }
}
//Loading this step's image.
SHOW image-viewer (img-iv)
//Colors counter.
SET histogram-counters (barcode-width,
                        barcode-height)
//Calculating the histogram.
SHOW histogram (histogram-counters)
END
```

ii. Histogram Equalization of the output image of Step ***i*** by spatially enhancement.

The following pseudocode illustrates this step:

```
//Barcode Area Enhancement Step ii
Pseudocode:
BEGIN
//Setting info for this step.
SET info
//Loop for the barcode's height.
FOR X=1 TO X=barcode-height STEP 1
{
```

```
  //Loop for the barcode's width.
  FOR Y=1 TO Y=barcode-width STEP 1
  {
    //Take pixel from img-iv.
    pixel=img-iv.GETPIXEL(Y,X)

    //The following two steps are used to
find
    //the index of pixel's color in the
    //grayscale table of 16 colors, then
    //increment the variable "no. of
    //pixels/color" by one...
    index = FIND-INDEX grayscale-table
[pixel]
    INCREMENT Nk [index] BY 1
  }
}

//Calculate the full barcode area size...
barcode-size = barcode-width * barcode-height

//Calculate the Probability Density
Function...
FOR X=1 TO X=16 STEP 1
  Pr[X] = Nk[X] / barcode-size
//Calculate the Transformation Function...
FOR X=1 TO X=16 STEP 1
{
  FOR Y=1 TO Y=X STEP 1
    S[X] = S[X] + Pr[Y]
}
//Normalize the Transformation Function to
//represent colors...
FOR X=1 TO X=16 STEP 1
  S[X] = S[X] * 255
//Now, Histogram Equalization...
//Loop for the barcode's height.
FOR X=1 TO X=barcode-height STEP 1
{
  //Loop for the barcode's width.
  FOR Y=1 TO Y=barcode-width STEP 1
  {
    //Take pixel from img-iv.
    pixel=img-iv.GETPIXEL(Y,X)
    //Find the pixel's color index from the
    //grayscale table and assign the
processing
    //pixel to the normalized color by the
    //found index (i.e. Normalized
    //Transformation Function)...
    index = FIND-INDEX grayscale-table
[pixel]
    pixel = S[index]
    //Put pixel in this step's image.
    img-vi.SETPIXEL(pixel,Y,X)
  }
}
//Loading this step's image.
SHOW image-viewer (img-vi)
//Calculating the histogram.
SHOW histogram (histogram-counters)
END
```

iii. Threshold the resultant image of Step ***ii*** at 112 grayscale color (many grayscale levels are taken and by experiment, it is found that the suitable threshold level is at 112.

The following pseudocode illustrates this step:

```
// Barcode Area Enhancement Step iii
Pseudocode:
BEGIN
//Setting info for this step.
SET info
//Loop for the barcode's height.
FOR X=1 TO X=barcode-height STEP 1
{
  //Loop for the barcode's width.
  FOR Y=1 TO Y=barcode-width STEP 1
  {
    //Take pixel from gray image.
    pixel=img-v.GETPIXEL(Y,X)

    //Threshold pixel at 112.
    IF pixel >= 112 THEN
        //Converts to white color.
        pixel = 255

    //Converts to black color.
    ELSE pixel = 0

    //Put pixel in this step image.
    img-vi.SETPIXEL(pixel,Y,X)
  }
}
//Loading this step's image.
SHOW image-viewer (img-vi)

//Calculating the histogram.
SHOW histogram (histogram-counters)
END
```

G. Decoding barcode area

After processing the captured image (barcode inside the image) throughout all the previous filters, now it is ready to be decoded. The decoding routine is implemented depending on the type of the barcode (symbology), the scanning directions are from left-to-right or vice versa "Omni-direction. After that the barcode number is transferred to the next operation

The following pseudocode illustrates this step:

```
// Decoding Barcode Area Pseudocode:
BEGIN
//Setting the barcode symbologies, direction
//scan, and how many the scanning line...
//The default barcode symbology.
SET BARCODE-TYPE = EAN-13
//The default scanning direction.
SET SCAN-DIRECTION = Left-to-Right
//From 1 to 20, 9 is the optimum number
(tested //experimentally).
SET SCANLINE-NUMBER = 9
//Start scanning within img-vi...
FOR X=1 TO X=SCANLINE-NUMBER   STEP 1
  scan-line[X] = SCAN (SCAN-DIRECTION, img-
vi)
//Averaging the number of the scanning...
FOR X=1 TO X=9   STEP1
  swap = swap + scan-line[X]
swap = swap / 9
//Now, decoding the averaged result, for more
//information about the barcode decoding
//algorithms see Appendix A...
barcode-number = DECODE-BARCODE(BARCODE-TYPE,
swap)
//Transfer the barcode number to the next
//section...
TRANSFER barcode-number
END
```

H. Display the resulted barcode number

The barcode number is displayed on the PC screen.

The following pseudocode illustrates this step:

```
// Display the Resulted Barcode No.
Pseudocode:
BEGIN
//Setting info for this operation.
SET info
//Displaying the barcode number on the img-
vi.
DISPLAY barcode-number ON img-vi
END
```

I. For new requested barcode processing

There is a decision operation (Node = New Barcode?) which take place after the whole operations and before the termination of the software. It is a custom selection for the end-user to decide either to restart the program from the beginning or close it.

The following pseudocode illustrates this step:

```
//Restart or Exit the software Pseudocode:
BEGIN
//Here, the end-user is ready to choose:
done = new-barcode-button-click
IF done = TRUE  THEN
  //either restart the software.
  GOTO restart
UNINITIALIZE directx-objects
UNINITIALIZE directx-filters
//or exit it.
EXIT
END
```

III. RESULTS

In this section, the software GUI is illustrated with the aid of four examples of different barcode symbologies (EAN-13, ISBN and UPC-A). The previous subsections of "II–System Layout " are illustrated completely using pseudocodes, while the four examples in this section are presenting only the captured image (barcode image) and the decoded one (displaying the decoding barcode number) results. The other steps of operations of the whole software are shown in the main GUI window as sequences.

A. EAN-13 barcode example

Captured barcode image via USB Webcam is shown in Figure 3.

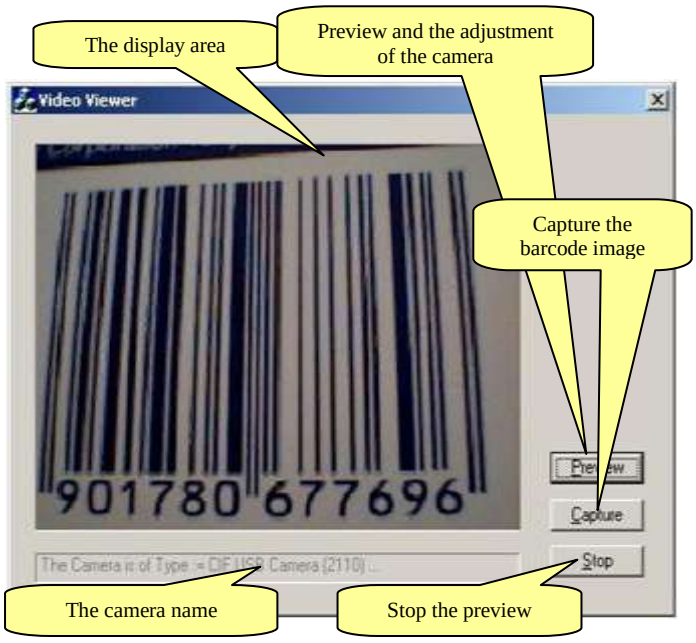

Fig. 3 Captured barcode via Webcam

Decoding barcode area and displaying the resulted barcode number are shown in Figure 4.

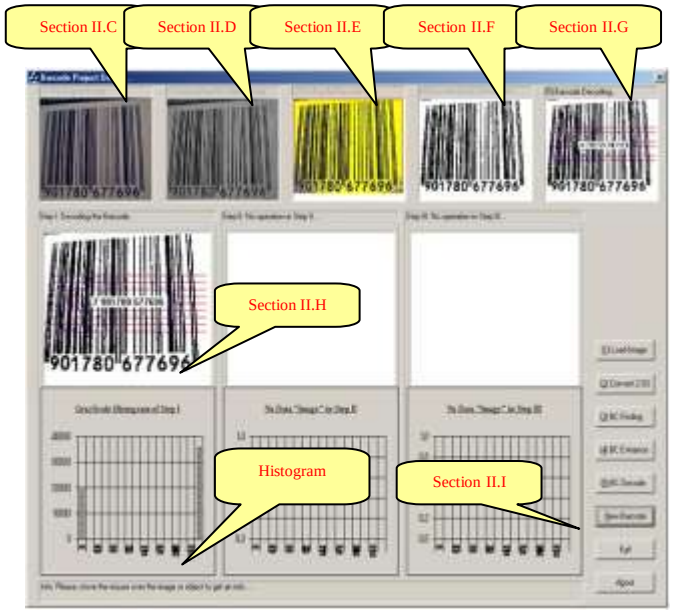

Fig. 4 Decoding barcode area and displaying the resulted barcode number

B. UPC-A barcode example

Webcam captured barcode shown in Figure 5.

Fig. 5 Capturing barcode via USB webcam

Decoding barcode area and displaying the resulted barcode number are shown in Figure 6.

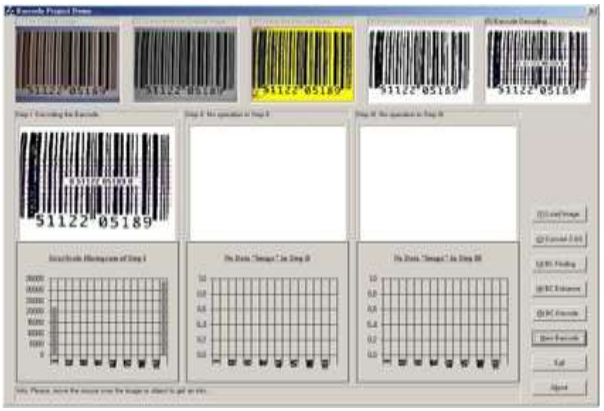

Fig. 6 Decoding barcode area and the resulted barcode number

C. 180^{o} ISBN barcode example

Example of captured inverted barcode is shown in Figure 7.

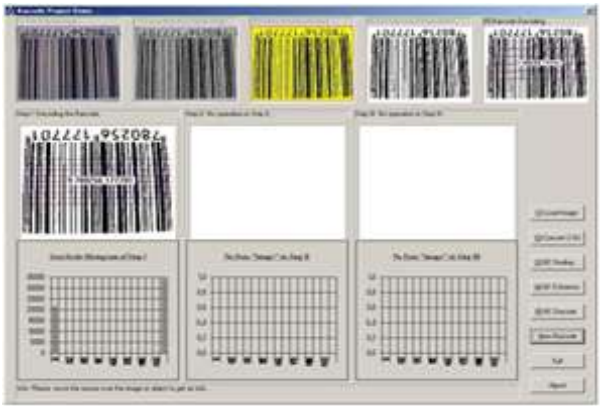

Fig. 7 Captured inverted barcode via webcam

Decoding barcode area and displaying the resulted barcode number is shown in Figure 8

Fig. 8 Decoding barcode area and resulted barcode number

D. Rotated ISBN with 5^{o} barcode example

Capturing barcode via USB Webcam, as shown in Figure 9.

Fig. 9 capturing barcode via USB Webcam

Decoding barcode area and displaying the resulted barcode number, as shown in Figure 10.

Fig. 10 Decoding barcode area displaying the resulted barcode number

IV. Discussion

A. Software GUI

When software starts running, the first panel appearing to the end-user is the "Video Viewer" window, as shown in Figure 3. The resultant of this viewer is the captured image, which it is a barcode image. And depending on the way of the capturing (like the justifying and calibration), this image will be the reference to the proceeding sections and will effect on the barcode decoding section.

The second panel of the software GUI is started with loading the captured image (barcode image) as shown in Figure. 4. The loaded image is displayed twice: the first one is the resized version of the loaded image which will not be changed during the next operations and the other one is processed according to the calculations and the appliance of digital image processing filters.

The histograms of the Red, Green and Blue channels are found and displayed in separated three histogram viewers.

Actually, there are no real digital image operations in this stage except that of the resizing and finding the histograms, but this stage actually prepares the captured image to the next digital image processing filters by finding its width and height.

The grayscale conversion is applied to the loaded image of the previous section (of the RGB Format "24-bit Color Quality"), to get an output image (of the Grayscale Format "4-bit Color Quality.

The purpose of this section is to grayscale the barcode image. So that, the easy manipulations for the grayscaled image (16 shades of gray color) than the colored one (different varicolored) for the next sections.

The grayscaled barcode image is ready for decoding, before that, it is efficient to find the location of the barcode area. According to customized digital image processing filters, the barcode area is filled with yellow color and surrounded with a colored rectangle, as shown in Figure 4.

The enhancement filters are applied to the founded barcode area by using the conventional digital image processing techniques. Now, the barcode image is ready for the decoding section.

The final barcode image is decoded and the resultant barcode number is displayed and printed on the barcode, as shown in Figure 4.

B. Software execution time

The average execution time of the software GUI (the average time is calculated for the four presented examples) are represented in Table 1.

V. Conclusions

1. The software capable of decoding the three barcode symbologies (EAN-13, ISBN and UPC-A) to a human-readable number or digits which represents different meaning according to that symbology and display the result on the PC monitor.
2. The capturing device driver interface works properly with the TwinkleCam USB Webcam. Thus, the software can extract the barcode area from the captured image successfully than other USB capturing devices.
3. The software has two methods of loading the barcode image: either from the capturing device directly or from the media storage device.
4. The purpose of grayscaling the barcode image is to unify the variations of color levels to defined set of grayscale levels. This grayscaling is depending on the look-up table of 16 shades of the gray.
5. The enhancement of the barcode image was based on the histogram-equalization method. The other methods or filters of the digital image enhancements were used like: low-pass, median and high-pass filters, but the convenient one is the histogram-equalization filter. Because it can keep some features that is necessary in the barcode image decoding.
6. The software may take long time to accomplish the whole operations from the loading the barcode image till decoding it. Because the software reads and collect its information of the processing images from the device context (i.e. monitor) and not from the image files.

Table 1
The average execution time for each section

Process (Section)	Step(s)	Execution Time (second)
Loading Barcode Image	1	1.789588
Gray scaling the Barcode Image	1	1.201759
Finding Barcode Area	3	3.105104
	Step *i*	0.647608
	Step *ii*	1.228748
	Step *iii*	1.716367
Barcode Area Enhancement	3	2.623505
	Step *i*	0.484292
	Step *ii*	1.253921
	Step *iii*	0.885292
Decoding Barcode Area & Displaying the Resulted Barcode Number	1	1.510932
Total Execution Time (second)		**10.230888**

REFERENCES

[1] Jeff Sloan, "Batch Capturing of Still Frames Images from Video", http://uas.usgs.gov/pdf/imageProcessing/ Raven_ Image _ Mosaic_Procedures.pdf
[2] Tanya H. Peters, M.Sc. Thesis "Effects Of Segmentation Routine And Acquisition Environment On Iris Recognition", University of Notre Dame, 2013.
[3] https://www.packagingcosmetics.com/wp-content/uploads /2014/07/ Barcode-Standards.pdf
[4] https://docs.isy.liu.se/twiki/pub/VanHeden/DataSheets/ ean13.pdf
[5] R.C. Gonzalez, "Digital Image Processing", 3d Edition.