Computer Vision project

Linear barcodes verification

Enrico Pittini (enrico.pittini@studio.unibo.it)
GitHub repository: https://github.com/EnricoPittini/Barcodes-verification

Contents

1 Introduction

The task consists in the verification of linear barcodes print quality according to ISO/IEC15416 specifications, using Image Processing and Computer Vision techniques.

The document `Linear-Barcodes-Verification-Project.pdf` contains the description of the project, while the document `guide-barcode-verification.pdf` contains an in-depth description of linear barcodes print quality verification.

For a more practical view, check out the doc-strings inside the code.

The *Python* [1] programming language has been used, with the following libraries: *NumPy* [2], *SciPy* [3], *OpenCV* [4], *Matplotlib* [5], *Pandas* [6].

In this report, several examples will be shown. They will mainly be about the image 'UPC#01'.

1.1 Description of the problem

Given an image containing a barcode, the task consists in verifying the print quality of the barcode, by computing some quality parameters.

Since the barcode in the input image can be rotated and can have different scales, and since the input image can contain other objects apart from the barcode, the quality parameters must be computed on a standardized sub-image. We refer to this image as *"refined ROI image"*, because it perfectly fits the Region Of Interest (i.e. the barcode) and it is refined according to some standards. More specifically, the refined ROI image is the sub-image of the input image which has the following properties.

- It contains the barcode, and the bars are perfectly vertical.
- Along the width, there are exactly $10 * X$ pixels before the first barcode bar and after the last barcode bar, where X is the minimum width of a bar (in pixels).
- Along the height, it perfectly fits the bar with smallest height. Basically, the height of the refined ROI image is equal to the minimum height of a barcode bar.

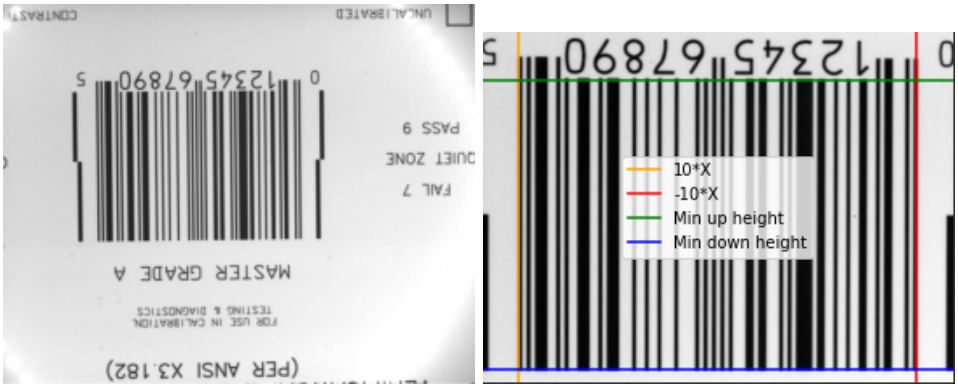
Figure 1 shows an example of ROI image, for the image 'UPC#01'.
Then, the print quality parameters are computed. For computing the quality parameters, N equally spaced horizontal scanlines are considered in the refined ROI image: by default, $N = 10$. Figure 2 shows an example, on the same image of before. The quality parameters are computed one each scanline, by considering its *scan reflectance profile*, and they are the following.

- Minimum Reflectance, i.e. $R_{\min}$.
- Symbol Contrast, i.e. SC. For computing it, also the Maximum Reflectance, i.e. $R_{\max}$, is taken into account.
- Minimum Edge Contrast, i.e. $EC_{\min}$.
- Modulation, i.e. M.
- Defect, i.e. D. For computing it, also the Maximum Element Reflectance Non-uniformity, i.e. $ERN_{\max}$, is taken into account.

Figure 3 shows an example of the scan reflectance profiles, with the quality parameters.

For each of these parameters, a numerical value is computed, and a symbolic grade between 'A' and 'F' is assigned, by using specific rules ('A' means very good, 'F' means very bad). In addition, a symbolic grade and a numerical value are assigned to the whole scanline.

Finally, an overall numerical value and an overall symbolic grade are assigned to the whole barcode.



(a) (b)

(c)

Figure 1: (a) Original image (b) Refinement of the ROI image (c) Refined ROI image

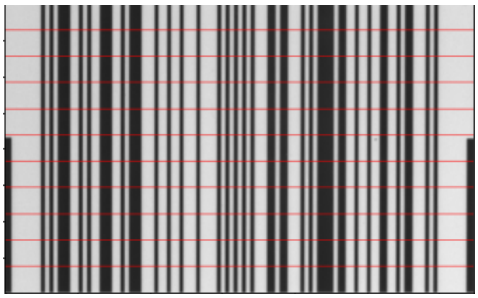


Figure 2: Refined ROI image with the 10 scanlines

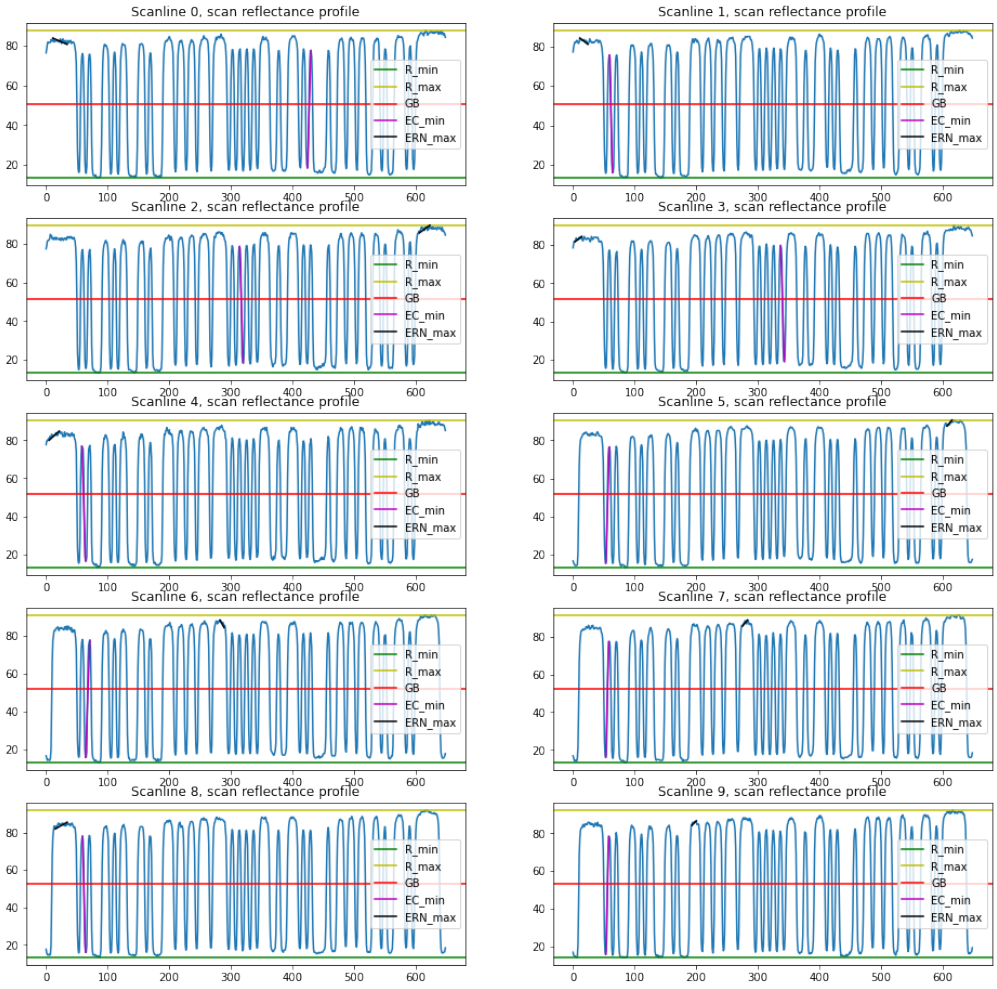
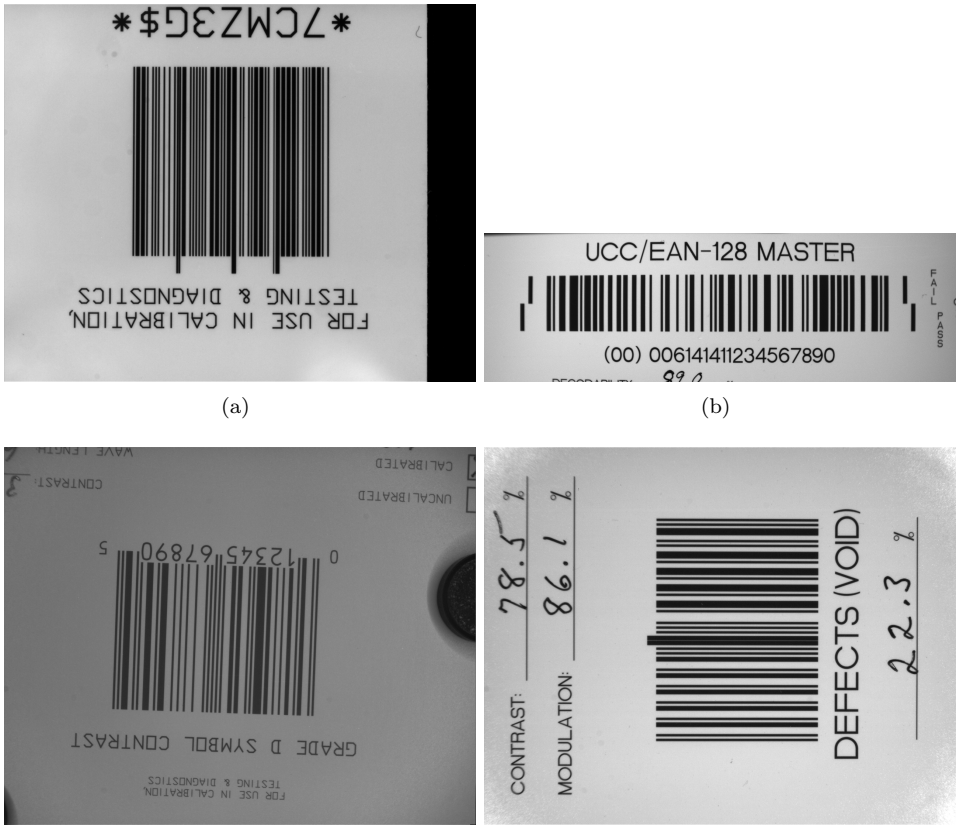


Figure 3: Scan reflectance profiles of the 10 scanline, with the quality parameters



(a) (b)

(c) (d)

Figure 4: Examples of some images in the given dataset

1.2 Dataset

Set of images containing barcodes, on which the application can be tested. Also an excel file containing the true quality parameters of these images is present. This dataset has been provided by *DATALOGIC*. A general overview of the dataset is now given.

Different kinds of barcodes are present. In addition, there are images in which the barcode is particularly rotated or scaled. There are also images in which the barcode has the bars which are horizontally aligned instead of vertically. Figure 4 shows some examples.

There are images in which the contrast, i.e. SC, is particularly bad on purpose. Figure 5 shows some examples.
Five images are particularly interesting for SC, since its grade ranges from 'A' to 'F': images 'UPC#03', 'UPC#04', 'UPC#05', 'UPC#06', 'UPC#07'.

There are also images in which the modulation, i.e. M, is particularly bad on purpose. For obtaining a bad M, an artifact has been added to the barcode, whose purpose is to decrease the intensity change between a bar and a space. Figure 6 shows an example.
Five images are particularly interesting for M, since its grade ranges from 'A' to 'F': images 'UPC#08', 'UPC#09', 'UPC#10', 'UPC#11', 'UPC#12'.

Finally, there are also images in which the defect, i.e. D, is particularly bad on purpose. For obtaining a bad D, a "fake vertical bar" has been added to the barcode, either inside a space or a bar, for increasing the range of different intensities inside that barcode element. This "fake vertical bar" is called *defect*, and it can be either inside a space, i.e. *defect spot*, or inside a bar, i.e. *defect void*. Figure 7 shows some examples.

(a) (b)

Figure 5: Examples of bad contrast images

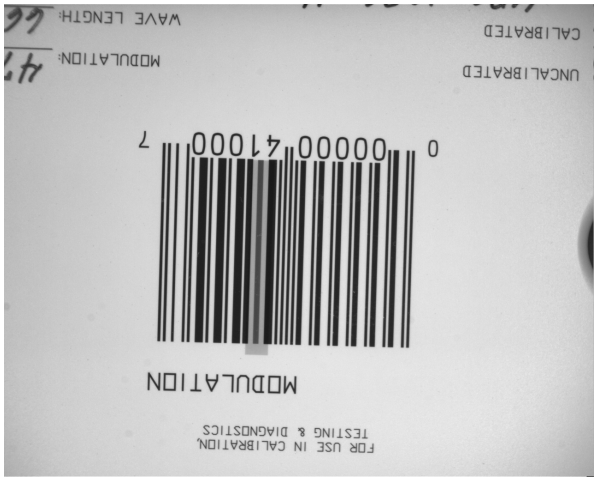
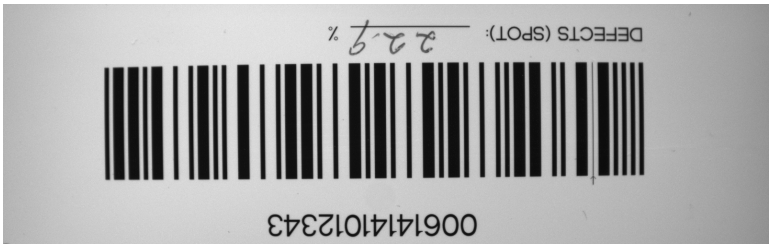


Figure 6: Example of a bad modulation image



(a)

(b)

Figure 7: Examples of bad defect images: (a) contains a defect spot, while (b) contains a defect void

Figure 8: Original image with the bounding box surrounding the barcode

Figure 9: Rotated image with the rotated bounding box

Five images are particularly interesting for D, since its grade ranges from 'A' to 'F', due to a defect spot: images 'UPC#13', 'UPC#14', 'UPC#15', 'UPC#16', 'UPC#17'.
Other five images have D which ranges from 'A' to 'F', but due to a defect void: images 'UPC#18', 'UPC#19', 'UPC#20', 'UPC#21', 'UPC#22'.

1.3 Approach

For solving our problem, a process consisting in four subsequent operations is implemented.

1.3.1 Bounding box detection

The bounding box surrounding the barcode in the input image is detected. Figure 8 shows the results on the image 'UPC#01'.

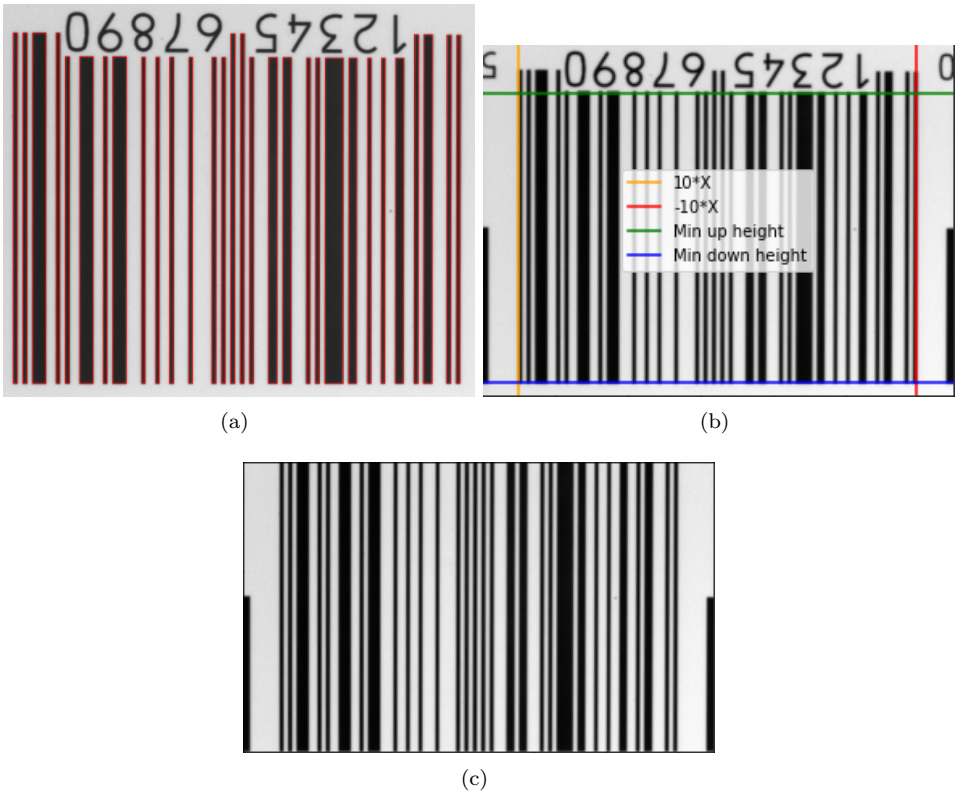
1.3.2 Bounding box rotation

The image and the bounding box are rotated in order to have the barcode bars perfectly vertical. From this operation, the ROI image is computed, which is the sub-image containing the barcode, with the bars perfectly vertical. Basically, the ROI image is the rotated image cropped around the rotated barcode. *Remark: the ROI image is gray-scale.*
Figure 9 shows the results, on the same image.

1.3.3 ROI image refinement

The ROI image is refined, according to the standard explained before.

- Along the width, the refined ROI image is such that there are exactly 10*X pixels before the first barcode bar and after the last barcode bar.



(a) (b)

(c)

Figure 10: (a) Barcode structure (b) Refinement of the ROI image (c) ROI image refined

- Along the height, the ROI image is refined in order to perfectly fit the bar with smallest height.

In order to perform this refinement, the precise and complete structure of the barcode is computed: every dimension about each bar is computed.
Figure 10 shows the results, on the same image.

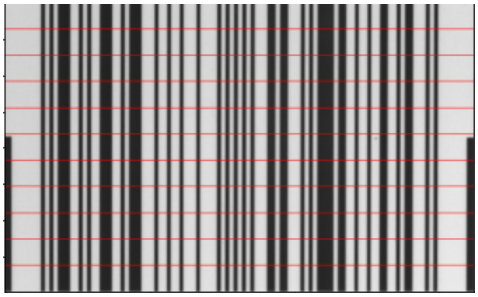
1.3.4 Computation of the quality parameters

The quality parameters of the barcode are computed, on the refined ROI image. As explained before, the following quality parameters are computed on each scanline:

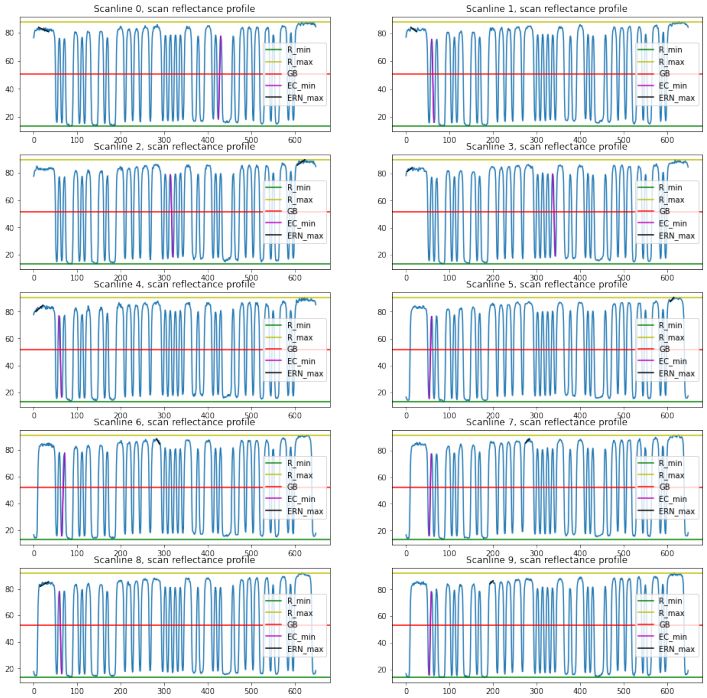
- $R_{\min}$, with also its symbolic grade
- SC, with also its symbolic grade
- $EC_{\min}$, with also its symbolic grade
- M, with also its symbolic grade
- $ERN_{\max}$, with also its symbolic grade
- D, with also its symbolic grade
- Symbolic grade and numerical value for the entire scanline

Figure 11 shows the results, on the same image.

Finally, an overall numerical value and an overall symbolic grade are assigned to the whole barcode.



(a)



(b)

Figure 11: (a) ROI image with the scanlines (b) Scan reflectance profiles, with the computed quality parameters

1.4 Output file

According to the project description, an excel output file must be generated, containing the information and results of the applied process.

In particular, the information is structured in the following sheets.

- **Global quantities.** It contains the following information.
  - Name of the image
  - Bounding box coordinates
  - Centre of the bounding box
  - Angle of the rotation
  - X dimension (i.e. minimum width of a barcode bar)
  - Height of the barcode (i.e. minimum height of a barcode bar)
  - Overall symbolic grade of the barcode
- **Bars/spaces widths** For each bar and space, its width is reported, in units by X dimension. It is a list, where the first element refers to the first bar, and the last element to the last bar. Basically, sequence of bars/spaces from left to right.
- **Scanlines quality parameters** For each scanline, its quality parameters are reported. Namely:
  - $R_{\min}$ and its symbolic grade
  - SC and its symbolic grade
  - $EC_{\min}$ and its symbolic grade
  - M and its symbolic grade
  - D and its symbolic grade
  - Symbolic grade and numerical value

Actually, this is the basic output format. The user can specify to build a richer output file, containing more information.

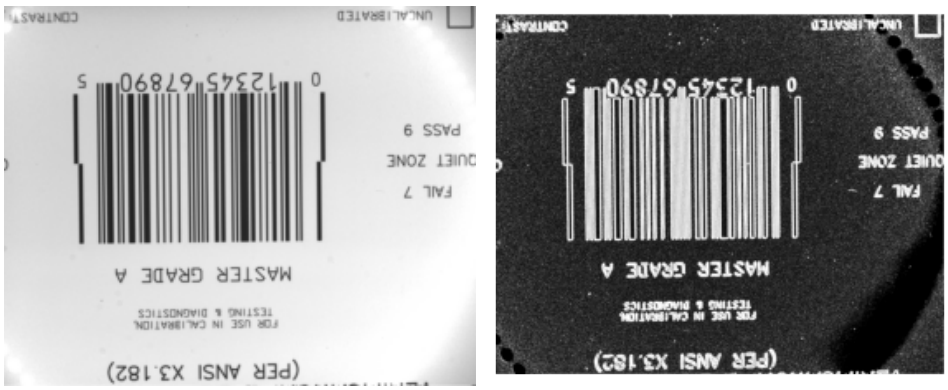
2 Bounding box detection

The detection of the bounding box surrounding the barcode mainly consists in thresholding the image and in applying a sequence of morphological operators. Let's inspect this process more in depth.

2.1 Computation of the derivatives

First of all, we want to highlight the pixels in which the bars are present. Basically, the binary thresholding is not applied on the original input image, but it is applied on a new image in which the bars pixels are pointed out (i.e. they have higher values). In this way, the thresholding operation is more reliable and effective. For pointing out the bars pixels, the derivatives are computed.

The contours of a bar can be seen as an image element in which there is a big change along a direction, and a small change along the perpendicular direction. If the bar is vertical, there is a big change along the horizontal axis and a small change along the vertical one; if the bar is horizontal, there is a big change along the vertical axis and a small change along the horizontal one. Therefore, if we compute the horizontal partial derivative and the vertical partial derivative images, we then subtract these two images and we finally take the absolute values, we obtain an image in which the bars contours are highlighted, both for vertical and horizontal bars. For simplicity, this final image is referred to as *gradient image*. The *Scharr* [7] operator is used for computing both the horizontal and vertical partial derivatives. Figure 12 shows an example.



(a) (b)

Figure 12: (a) Original image, image 'UPC#01' (b) Gradient image

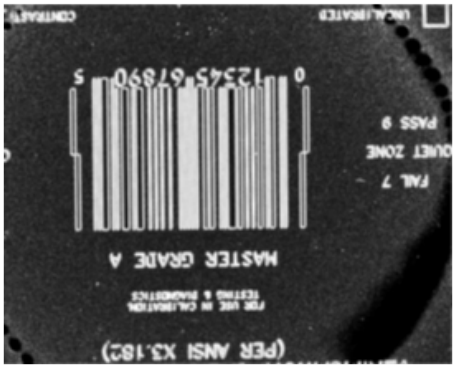
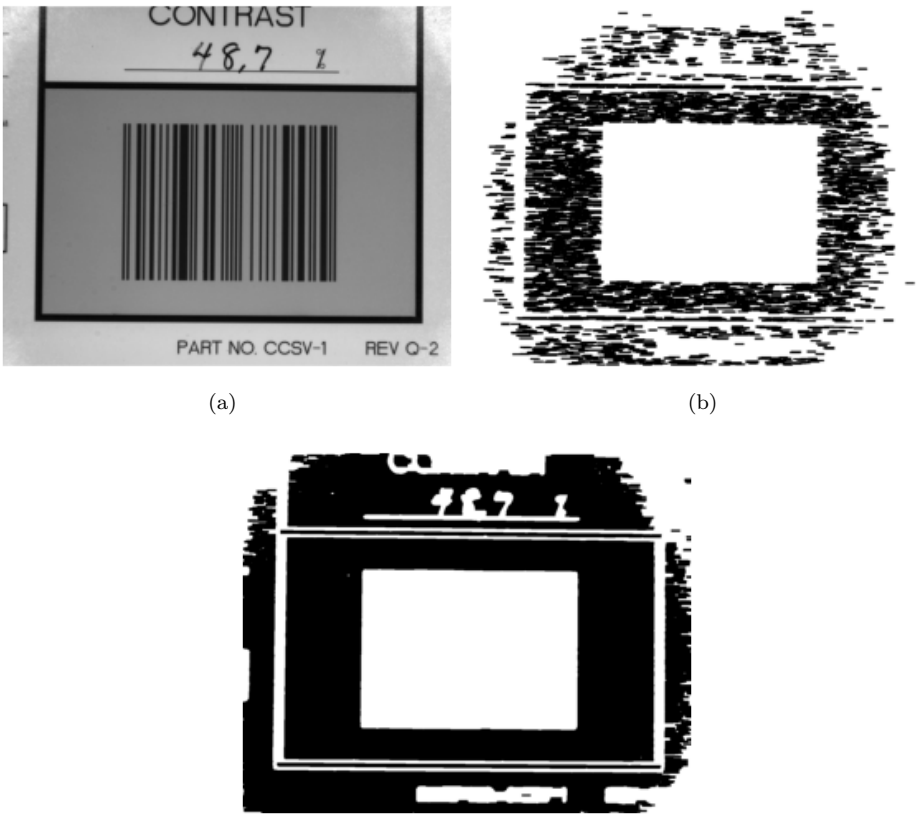


Figure 13: Gradient image after the denoising, image 'UPC#01'



(a) (b)

(c)

Figure 14: (a) Original image (b) Results after the closing operation with the non-blurred gradient image (c) Results after the closing operation with the blurred gradient image

2.2 Denoising

The gradient image is blurred, using mean filtering. Figure 13 shows an example. This denoising is useful for improving the results of the closing operation, which will be applied later, after the thresholding. Thanks to the denoising of the gradient image, the results after the closing operation are much better: the barcode region is much more separated and clear. Figure 14 shows an example, on the image 'EAN-UPC-CONTRAST IMGB'.

2.3 Thresholding

The blurred gradient image is now thresholded: this resulting binary image is referred to as *thresholded image*. Using the Otsu's algorithm, a threshold is chosen, for performing a foreground-background segmentation. Since this is applied on the gradient image, only the pixels which have an high partial derivative along an axis and a low partial derivative along the perpendicular axis are kept as foreground. Ideally, these should only be the bars contours pixels: but, of course, not only bars pixels are detected as foreground, but also pixels of many other image entities.
On the whole, there are both kinds of errors.

- False Negatives, i.e. barcode pixels detected as background. This kind of error is due to the fact that only bars contours have high values in the gradient image, and not all barcode pixels
- False Positives, i.e. non-barcode pixels detected as foreground. This kind of error is due to the fact that also other image entities can have high change along one axis and low change along the perpendicular one.

Figure 15 shows an example.

Figure 15: Thresholded image, image 'UPC#01'

Figure 16: Closed image, image 'UPC#01'

2.4 Closing

The closing operator is applied on the thresholded image: the resulting image is referred to as *closed image*. The aim is to solve the False Negatives problem: we want to 'fill' the barcode foreground region. Basically, we want to be sure that all barcode pixels are now detected as foreground.
This is achieved by means of a closing operator with a rectangular structuring element of size 25×25. Figure 16 shows an example.

Choosing the right size of the structuring element is really important.

- If the size is too small, then it can happen that not all barcode pixels are detected as foreground. Basically, the barcode region is not fully detected as foreground. In particular, this happens if the structuring element fit in a background space between bars inside the barcode region. Figure 17 shows an example, on the image 'EAN128-CONTRAST IMGB'.
- If the size is too big, then it can happen that too many external non-barcode elements are put inside the barcode region. This can be bad because it can alter the correct detection of the bounding box.

2.5 Opening

The opening operator is applied on the closed image: the resulting image is referred to as *opened image*. The aim now is to decrease the False Positives problem: we want to clean, as much as possible, the foreground pixels outside the barcode region.
This cleaning is achieved by means of an opening operator, with a rectangular structuring element of size 50×50. Figure 18 shows an example.

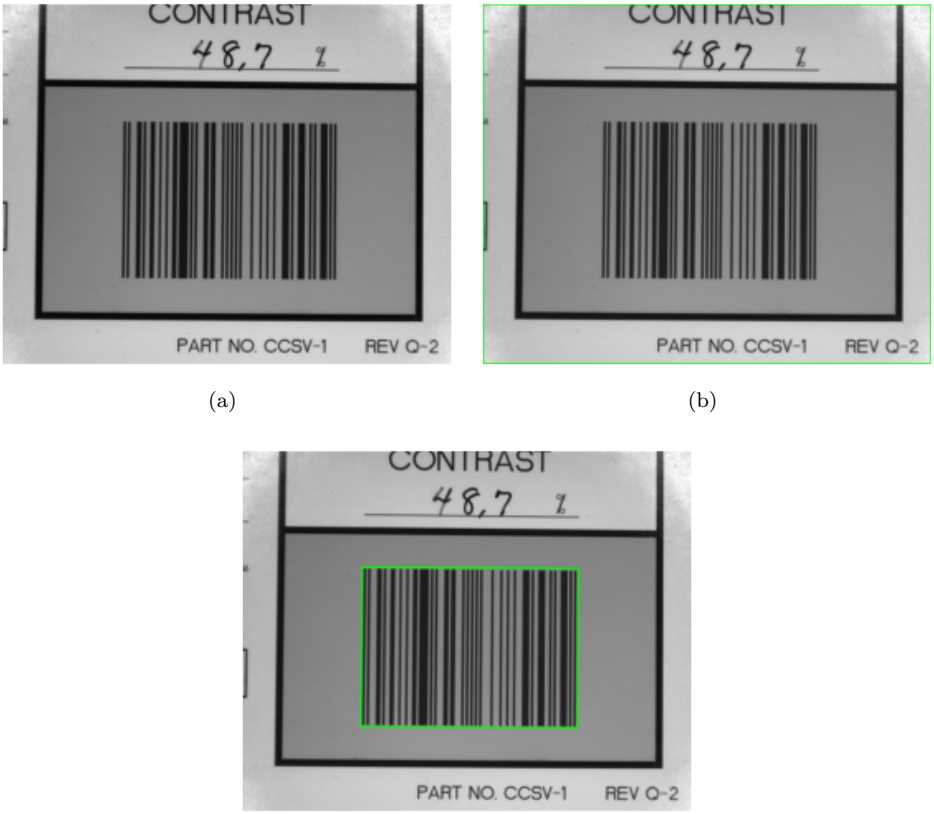
This removal of foreground entities outside the barcode region is important for two reasons.

(a) (b)

(c)

Figure 17: (a) Original image (b) Closing with a too small structuring element (c) Closing with a big enough structuring element

Figure 18: Opened image, image 'UPC#01'



(a) (b)

(c)

Figure 19: (a) Original image (b) Bounding box obtained with an opening with a too small structuring element (c) Bounding box obtained with an opening with a big enough structuring element

- First of all, since the image is cleaner, it simplifies and improves the bounding box detection. For achieving this, choosing the right size of the structuring element is really important: for instance, if too small, the detected bounding box can end up in being too large. Figure 19 shows an example, on the image 'EAN-UPC-CONTRAST IMGB'.
- Moreover, since external foreground entities can be near the barcode region, they can negatively affect the orientation of the barcode region. Therefore, by removing them, the barcode orientation is better estimated. For achieving this, choosing the right size of the structuring element is really important: for instance, if too small, the detected bounding box can end up in having a clearly wrong orientation. Figure 20 shows an example, on the image 'I25-DEFECTS IMGB'.

2.6 Dilation

As last morphological step, the dilation operator is applied on the opened image: the resulting image is referred to as *dilated image*. A very low dilation operation is applied: structuring element of size 8×8.
The purpose of this operation is to slightly enlarge the barcode foreground region in order to be sure that all barcode pixels, even the ones in the very contours of the barcode region, are correctly identified as foreground. Indeed, we want to be absolutely sure that the whole barcode is fully contained in our bounding box, even the first and last bars, because later we want to compute the complete structure of the barcode. Therefore, it is better to have a slightly bigger bounding box than a smaller one.
Figure 21 shows an example.

Of course, also here, choosing the right size of the structuring element is really important. We don't

(a) (b)

(c)

Figure 20: (a) Original image (b) ROI image obtained with an opening with a too small structuring element (c) ROI image obtained with an opening with a big enough structuring element

Figure 21: Dilated image, image 'UPC#01'

Figure 22: Detected bounding box, image 'UPC#01'

want to enlarge the barcode region too much, because this could affect the shape of the region, and thus affect the correct orientation of the barcode.

2.7 Bounding box

The dilated image is a binary image in which the biggest and clearest foreground region is the barcode region. Now, the bounding box enclosing that region is found.
For achieving this, the following steps are performed.

1. The contours vertices of each separated foreground region are found. So, list of contours vertices, one for each different foreground region.

2. This list is sorted according to the area of the foreground regions. Then, only the first element of this list is taken: this represents the contours vertices of the barcode region.

3. Finally, the smallest bounding box enclosing these contours vertices is found. This is our final bounding box.

Figure 22 shows an example.

2.8 Final comments

This just described bounding box detection pipeline works pretty well with all the dataset images. The detected bounding box fully contains the barcode, with the right orientation, while keeping outside external non-necessary elements. In addition, this method works well with barcodes with different orientations, scales and contrasts. Figure 23 shows some examples on dataset images. This method has also been tested with other external barcode images, obtaining good results.

However, this bounding box detection pipeline has some drawbacks.

- First of all, the gradient image consists in highlighting the pixels which have high change along one axis and low change along the opposite one. This makes sense for pointing out the bars contours pixels, but only if the barcode is not too much rotated. In particular, if the barcode angle is exactly 45°, as the barcode in image 24 (which is not a dataset image), our method does not work. So, our method works well with barcodes with different orientations, but only if they are not too much rotated, like having a 45° angle.

- Then, the biggest problem is about the sizes of the structuring elements of the morphological operators. Indeed, these sizes have been chosen by means of a try-and-error approach on the dataset images. So, they have been carefully fixed by hand for obtaining good results on the specific given images. Therefore, our method could work not well on other images, especially if they are pretty different from the dataset images (e.g. image which contains a small barcode and a lot of textual elements around it).

(a) (b) (c) (d) (e)

Figure 23: Examples of some dataset images (a) Image 'C128_75LOW' (b) Image 'EAN-UPC-UPC-A MASTER GRADE IMGB' (c) Image 'EAN128-MASTER IMGB' (d) Image 'UPC#07' (d) Image 'UPC#10'

Figure 24: 45° barcode

(a) (b)

Figure 25: (a) Original image 'UPC#01' (b) Rotated image

Nonetheless, it is important to point out that our dataset is quite diversified and comprehensive, meanining that it contains several different types of barcodes, with different orientations, scales and contrasts. Therefore, even if our method does not work well on all possible barcodes images, it works well with several different ones. Especially if we restrict to images such that the barcode bars are pretty much aligned with one of the two image axes: this is actually quite common, particularly in industrial settings.

3 Bounding box rotation

The image and the bounding box are rotated such that the barcode bars become perfectly vertical.

3.1 Orientation of the barcode

First of all, the orientation of the barcode is found, i.e. the angle with respect to the horizontal image axis.
This angle is simply estimated as the angle between the horizontal axis and the line connecting the left-up and the right-up bounding box vertices. More precisely, the angle is computed by considering the slope of that line connecting the left-up and the right-up bounding box vertices. If $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ are these two vertices, then the angle is computed as:

$$\alpha = \arctan \frac{y_2 - y_1}{x_2 - x_1}$$

3.2 Rotating the image

The input image is now rotated by this angle α, with respect to the left-up bounding box vertex. In such way, the barcode bars become perfectly vertical.
This rotation operation consists in applying a warping.
Figure 25 shows an example.

3.3 Rotating the bounding box

Since the rotation has been performed with respect to the left-up bounding box vertex, rotating the bounding box vertices is trivial. The left-up vertex remains the same, while the others are simply obtained by considering that the bounding box must be perfectly aligned with respect to the image axes and that it must have the same width and height of before. For instance, the right-up vertex is obtained by taking the point which has the same vertical coordinate of the left-up vertex and such that the distance between the two is equal to the bounding box width.
Figure 26 shows an example.

Figure 26: Rotated image 'UPC#01', with the rotated bounding box

Figure 27: ROI image, from image 'UPC#01'

3.4 ROI image

Finally, the ROI image is computed, which is the sub-image containing the barcode, with the bars perfectly vertical. Basically, the ROI image is the rotated image cropped around the rotated barcode.
Remark: the ROI image is gray-scale.
Figure 27 shows an example.

3.5 Horizontal bars case

All the operations performed since now are correct assuming that the barcode bars are vertical. If the bars are horizontal, our approach does not work as desired: the rotated image is such that the bars are perfectly horizontal, instead of perfectly vertical. Figure 28 shows an example, on the image 'EAN-UPC-DEFECTS IMGB'.

For fixing this situation, the following steps must be performed.

1. **Understand if the barcode bars are vertical or horizontal.** This is achieved by understanding if, in the ROI image, there are more horizontal changes or more vertical changes. More precisely, the horizontal partial derivative and vertical partial derivative images of the ROI image are computed, using the Scharr operator. If the sum over the absolute values on the latter image is bigger than the sum over the absolute values on the former image, then this implies that there are more vertical changes than horizontal ones in the ROI image, meaning that the barcode bars are horizontal.

2. **If the bars are horizontal, then rotate the image and the bounding box in order to have the bars perfectly vertical.** So, another rotation must be performed. This is achieved through a warping.

Figure 28: Rotated image, with the rotated bounding box

Figure 29: Rotated image, with the rotated bounding box

After these two steps, we are sure that the rotated image, the rotated bounding box and the ROI image contain the barcode with perfectly vertical bars.
Figure 29 shows an example, on the same image 'EAN-UPC-DEFECTS IMGB'.

4 ROI image refinement

The ROI image is refined, according to the standard explained in the introduction.

- Along the width, the refined ROI image is such that there are exactly 10*X pixels before the first barcode bar and after the last barcode bar.
- Along the height, the ROI image is refined in order to perfectly fit the bar with smallest height.

In order to perform this refinement, the precise and complete structure of the barcode is computed.
It is very important to point out from the beginning that the barcode structure is computed with respect to the ROI image: this is the reference system.

4.1 Computing the barcode structure

Both local and global quantities about the barcode structure are computed.

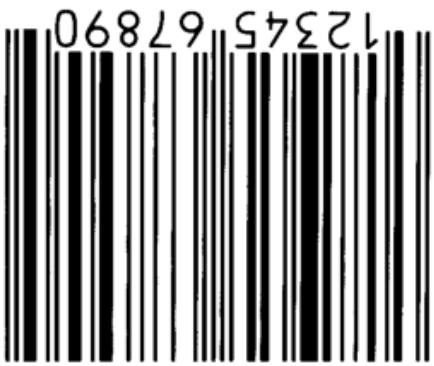


Figure 30: Thresholded ROI image, image 'UPC#01'

- Local quantities: quantities about the individual single bars. For each bar, the following are the local quantities.
  - Horizontal coordinate of the first pixel.
  - Width.
  - Half height up: half height from the middle of the ROI image upward.
  - Half height down: half height from the middle of the ROI image downward.
  - Height (computed by adding together the last two quantities)
- Global quantities: quantities about the whole barcode. They are computed from the local quantities. The global quanties are the following.
  - X: minimum width of a bar.
  - hhUp_{min}, i.e. 'min half height up': minimum half height up of a bar.
  - $\text{hhDown}_{\text{min}}$, i.e. 'min half height down': minimum half height down of a bar.
  - h: height of the barcode (computed by adding together the last two quantities)
  - fb_x, i.e. 'first bar x': horizontal coordinate of the first pixel of the first bar (i.e. left-most bar).
  - lb_x, i.e. 'last bar x': horizontal coordinate of the last pixel of the last bar (i.e. right-most bar).

Once the local structure of the barcode has been computed, computing the global structure is straightforward. Therefore, in the next section, the focus will be on the computation of the local structure.

4.2 Algorithm for computing the local structure

The basic algorithm is the following.

1. Threshold the ROI image, either using a new threshold (Otsu's algorithm) or using the same threshold used for the bounding box detection. *The user can specify the option to follow, but it is suggested to use a new threshold: it is more reliable, even if it is more expensive to compute.* The thresholded ROI image is a binary image in which the white pixels represent spaces pixels, while dark pixels represent bars pixels. Of course, as for all the thresholded binary images, there can be errors: bars pixels which are white and space pixels which are dark. In practice, a bar can have holes and it can goes upon the adjacent space. Figure 30 shows an example of thresholded image
2. Consider the horizontal line passing through the exact half height of the ROI image. Scan each pixel of this whole horizontal line, from left to right.

Figure 31: Wrong barcode structure, image 'UPC#01'

3. If space pixel, continue to the next pixel and repeat. If bar pixel, this means that it is the first pixel of a bar: go to step 4.

4. Continue to scan the pixels in the horizontal line, until finding a space pixel. This number of adjacent bars pixels is the width of this current bar. For computing the half height up, start from the first bar pixel and go up along that vertical line, until finding a space pixel: this number of adjacent bars pixels is the half height up of the current bar. Do the same thing also for computing half height down.
   After this step, the local quantities related to this current bar are computed.
   Then, continue the scanning along the horizontal line in the middle of the ROI image: go to the next pixel to the right after the last bar pixel, and repeat step 3.

It is worth making two observations.

- The scanning is performed along the horizontal line in the exact middle of the ROI image because it is the horizontal line along which it is the most probable to find all the barcode bars.

- The computation of half height up and down can be performed in a single cycle, for improving the performances: we go up and down at the same time.

Actually, this algorithm does not work. Because, as it has been said before, there can be errors. In particular, while going up/down vertically for computing the half height up/down, it can happen that we find a wrong white pixel, which should be a bar pixel: this is an 'hole' in the current bar. This implies that a smaller half height is computed for that bar. Figure 31 shows an example, in which the half heights up/down of most of the bars are wrong.

Improving this is actually quite simple. It is enough to make two little changes to the algorithm.

- The vertical exploration for computing half height up/down now starts from the pixel in the middle of the current bar, rather than from the first bar pixel. This improves a lot the goodness of the results, because most of the 'holes' in a bar (i.e. bar pixels which are white) are present in the border of the bar. Most of the 'holes' in a bar are on the border: they are degradated contours. This is evident by looking to the thresholded ROI image in figure 30.

- For further improving the reliability, when going up/down vertically along the middle vertical line in the current bar, a single white pixel is not anymore enough for stopping the exploration, but three horizontally-adjacent white pixels are needed. So, the pixel in the middle vertical line must be white, but also the next pixel to the left and also the next pixel to the right.

These two little changes improve a lot the situation: the algorithm now works correctly for all the dataset images. Figure 32 shows an example of correct barcode structure.

Figure 32: Correct barcode structure, image 'UPC#01'

4.3 Insight: other algorithms for computing the local structure

The current algorithm, which, from now on, will be called *algorithm* 1, works pretty well and it is pretty fast, but its ways of approximating the width and the half height up/down of a bar are pretty basic. Indeed, the width of a bar is approximated by simply considering only the dark pixels present in the horizontal line in the middle, while the half height up/down by simply considering only the middle vertical line. Since, as said before, there can be errors, both wrong white pixels and wrong dark pixels, one could be interested in trying to make this approach more sophisticated.

Therefore, three other different algorithms have been developed, more and more complex, but also more and more expensive. Despite the bigger complexity of these algorithms, they don't improve in a significant way the situation, at least in the dataset images. The basic algorithm 1 achieves results which are comparable with the results achieved by the most sophisticated algorithm (i.e. algorithm 4), while being much more faster. Therefore, it is recommended to use the first algorithm.

Nonetheless, these three algorithms are now briefly described. For having more details, check their implementation in the code, inside the `refine_ROIimage` python file.

- **Algorithm** 2
  Since there can be errors, computing the width of a bar by only considering one horizontal line (i.e. the one in the middle) can be problematic. In particular, we focus on the fact that white pixels could be wrong, and they should belong to the bar: therefore, our computed width could be an underestimation of the true width. Thus, the idea is to, while exploring vertically the bar for computing half height up/down, updating the width of the bar if new dark pixels are found in the border. In this vertical exploration, at each vertical level, the current width of the bar is increased by adding the number of adjacent pixels which are found next to the current bar borders (both left and right).
  Also the stopping criterion of the vertical exploration is made more sophisticated. The vertical exploration, either up or down, stops when the number of actual dark pixels among all the pixels which should be dark in that vertical level, according to the current bar width, is less than a certain percentage (e.g. 0.33).
  This algorithm is not too much more complex or expensive that the first one. In addition, it has a nice property: it ensures that, for each bar, no bar pixel remains outside the computed rectangle enclosing that bar. Basically, the computed rectangle it is an overestimation of that bar, which for sure contains all its pixels.
  However, this algorithm is actually worse than the first one, at least on the dataset images. Indeed, the weak point of this algorithm is that it considers as possible errors only the white pixels, but, actually, also dark pixels can be errors. Basically, a single wrong dark pixel adjacent to the bar contours is enough for enlarging the computed width. Figure 33 shows an example of that.

- **Algorithm** 3

Figure 33: Example of barcode structure computed with the algorithm 2, image 'EAN-UPC-UPC-A MASTER GRADE IMGB'

The weak point of algorithm 2 is solved by approximating the width of a bar with the mean of the widths found during the vertical exploration. Actually, the computation of the width is broken into the two halves, from the middle vertical line of the bar: left width and right width (this is done for making the approximation more robust). At each vertical level during the vertical exploration, starting from the pixel in the middle of the bar, the left width and the right widths of that level are computed. Then, the true left width and right width of that bar are computed by averaging the new computed ones with the ones already found in the vertical exploration.
The stopping criterion of the vertical exploration is the same of the one in algorithm 2.
Even if it is clearly a more robust approach, it has several drawbacks. First of all, it is way slower than the first algorithm. Then, since the mean computes floating point numbers, they must be rounded. Finally, the mean is sensible to outliers: therefore, if there is a single vertical level in which the bar width is completely wrong, this affects the computed width.

- **Algorithm** 4
  The algorithm 3 is improved by using the mode instead of the mean. Basically, the left width is computed as the most frequent left width of that bar, encountered during the vertical exploration. The same for the right width. The rest of the algorithm is the same.
  This is the most reliable and precise algorithm, but it is also the slowest one.

As said before, even the algorithm 4 does not improve much the results with respect to the basic algorithm 1. The reason of that is the fact that, as said in the previous section, the errors, either wrong white or wrong dark pixels, are mainly along the borders of a bar, and not in the middle. This means that the basic approach of performing the vertical exploration only along the single vertical line in the middle of the bar is not a bad idea. It is the best trade-off between precision and speed.
The following table shows the average solving time, in seconds, of the different algorithms across all the dataset images.

Alg1	Alg2	Alg3	Alg4
0.022	0.08	0.25	0.40

4.4 Removing the wrong bars

After having computed the local structure, and before computing the global structure, an intermediate step must be performed. Since now, we have assumed that inside the ROI image there are not other entities apart from the barcode bars. However, this is not necessarily true. Therefore, it can happen that the algorithm which computes the local structure wrongly identifies as a bar an element which is not a bar. Figure 34 shows two examples of that. Actually, these are the only images in which this particular thing happens. And it is worth observing that, in the second image, the wrong detected bar is actually a defect spot.

(a) (b)

Figure 34: Examples of two images in which there are wrongly detected bars.

So, an algorithm for identifying the possible wrong bars must be defined: then, after their identification, these bars are simply removed from the computed local structure.

The algorithm for identifying possible wrong bars is based on the idea that wrong bars are outliers with respect to the other detected bars. They are "strange and anomalous" bars with respect to the others. More specifically, they are outliers with respect to the bars height and area: their height and area is very small or very big compared to the other bars.

The algorithm used for detecting the outliers in a generic vector v is the following, which takes strong inspiration for the *classic statistical way of detecting outliers (used also in the boxplot)* [8]. The quantiles q_1, q_2 are computed, which have, respectively, level l and $1 - l$. This $l \in [0, 0.5]$ is fixed by the user. Then, the interquantile range IQR is computed as $IQR = q_2 - q_1$. Finally, a value in v is an outlier if its value is either smaller than $q_1 - IQR$ or bigger than $q_2 + IQR$.

The algorithm for detecting the wrong bars is the following.

1. **Detect the outlier bars with respect to the bars height, using level l.** If no outlier bar is detected, then end the algorithm: no wrong bar is present.
2. **Detect the outlier bars with respect to the bars height, using the same level l.** If no outlier bar is detected, then end the algorithm: no wrong bar is present.
3. The wrong bars are the bars which are outliers with respect to both height and area.

It is very important to ensure that no true bar is pruned from the local structure. This is the reason why a bar is considered wrong only if it is an outlier with respect to both height and area. In addition, this is also the reason of the two following considerations.

- It is suggested to use a pretty small value for l: the default one is 0.02.
- In the actual implementation in this repository, the algorithm detects at most one wrong bar. And, in particular, a bar is considered wrong if it is the most outlier bar with respect to both height and area.

Figure 35 shows the fixed local structure for the two images seen before.

4.5 Refining the ROI image

After having computed the local structure and after having possibly fixed it, the global structure is computed. Then, the ROI image can finally be refined, according to the standard.

First of all, the current bounding box (the one obtained from the rotation, i.e. the rotated bounding

(a) (b)

Figure 35: Fixed local structure for the two images

box) is refined according to the standard. Then, the refined ROI image is obtained by simply cropping the current input image (the rotated one) aroung the refined bounding box. Basically, once the bounding box coordinates are refined, obtaining the refined ROI image is trivial.

The refinement of the (rotated) bounding box works as follows.

- **Refinement along the width**
  The vertical line passing through the bounding box left-up and left-bottom vertices is moved such that there are exactly $10 * X$ pixels before the first bar. Basically, the horizontal coordinate of these two vertices is decreased by the quantity $10 * X - \mathrm{fb}_x$, where fb_x is the horizontal coordinate of the first pixel of the first bar.
  At the same time, the vertical line passing through the right-up and right-bottom vertices is moved such that there are exactly $10 * X$ pixels after the last bar. Basically, the horizontal coordinate of these two vertices is increased by the quantity $10 * X - (w - \mathrm{lb}_x - 1)$, where w is the width of the non-refined bounding box and lb_x is the horizontal coordinate of the last pixel of the last bar.

- **Refinement along the height**
  The horizontal line passing through the bounding box left-up and right-up vertices is moved such that if exactly fits the min half height up of the barcode. Basically, the vertical coordinate of these two vertices is increased by the quantity $hh - 1 - hhUp_{\min}$, where hh is the half height of the non-refined bounding box and $hhUp_{\min}$ is the min half height up of the barcode.
  At the same time, the horizontal line passing through the bounding box left-bottom and right-bottom vertices is moved such that if exactly fits the min half height down of the barcode. Basically, the vertical coordinate of these two vertices is increased by the quantity $hh - 1 + hhDown_{\min}$, where hh is the half height of the non-refined bounding box and $hhDown_{\min}$ is the min half height down of the barcode.

Figure 36 shows an example.

5 Computation of the quality parameters

The quality parameters of the barcode are computed, on the refined ROI image. As explained in the introduction, N equally spaced horizontal scanlines are considered: by default, $N = 10$. Figure 37 shows an example.

Then, the following quality parameters are computed on each scanline:

- Minimum Reflectance, i.e. $R_{\min}$.
- Symbol Contrast, i.e. SC. For computing it, the Maximum Reflectance, i.e. $R_{\max}$, is also taken into account.

(a) (b)

Figure 36: Image 'UPC#01': (a) Refinement of the ROI image (c) ROI image refined

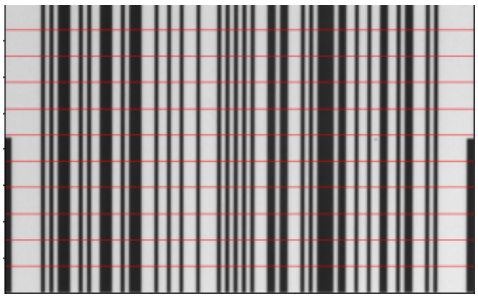


Figure 37: Scanlines for the image 'UPC#01'

- Minimum Edge Contrast, i.e. $EC_{\min}$.
- Modulation, i.e. M.
- Defect, i.e. D. For computing it, the Maximum Element Reflectance Non-uniformity, i.e. $ERN_{\max}$, is also taken into account.

A symbolic grade is assigned to each parameter. Then, a symbolic grade and a numerical value are assigned to the whole scanline.
Finally, an overall numerical value and an overall symbolic grade are assigned to the whole barcode.

5.1 Scan reflectance profile and $R_{\min}, R_{\max}, SC, GB$

For each scanline, its scan reflectance profile is taken into account, which is simply the curve of the percentage pixel intensities along that horizontal line.
$R_{\min}$ and $R_{\max}$ are simply the, respectively, min and max values along the scan reflectance profile.
SC is the difference between $R_{\max}$ and $R_{\min}$.
Finally, also the Global Threshold GB is computed: $GB = R_{\min} + SC/2$.
Figure 38 shows an example.

Among these quantities, only $R_{\min}$ and SC are quality parameters. The smaller is $R_{\min}$, the better is the barcode print quality. The bigger is SC, the better is the barcode print quality.

5.2 Edges, peaks and valleys

For computing the next quality parameters, the edges, peaks and valleys of the scan reflectance profile must be computed.

The **edges** are the scanline points which separate adjacent bars-spaces. They represent the change from a bar to a space, or the opposite. For discerning bars/spaces, the GB is used: a pixel whose

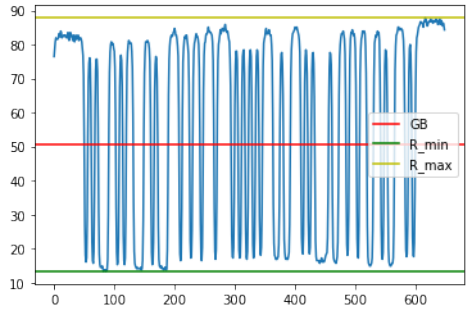


Figure 38: Scan reflectance profile, with the computed quality parameters, for the first scanline of the image 'UPC#01'

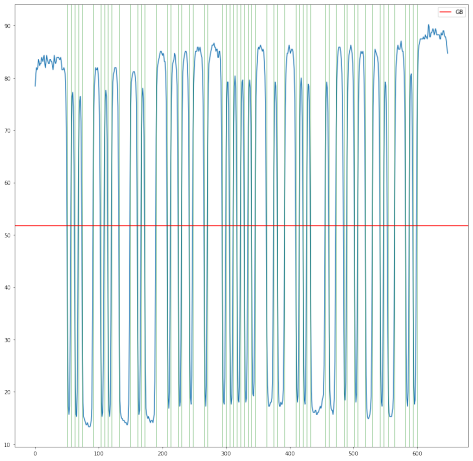


Figure 39: Edges of the first scanline of the image 'UPC#01'

value is less than GB is a bar pixel, a pixel whose value is greater than GB is a space pixel.
Remark: in computing the barcode structure, we have precisely detected the bars, while here a way more basic approach has been used. This basic approach is defined by the standard for computing the quality parameters. It can happen that, using this basic approach, the bars are not correctly detected: in such case, this means that the print quality of the barcode is not good, and, therefore, the quality parameters have bad values.
For detecting which scanline pixel is an edge, a basic approach has been used.

- Compute a boolean mask m: $m[i] = 1$ iff the scanline pixel i is a bar pixel.
- Shift m by one position to the left: mask m_1. Basically, the first pixel is deleted.
- Shift m by one position to the right: mask m_2. Basically, the last pixel is deleted.
- Compute the element-wise absolute difference $|m_1 - m_2|$. This is our final mask, which says which are the edge pixels in the scanline. Actually, for making that final mask to have the same length of the scanline, a 0 value must be added at the beginning: the first scanline is for sure not an edge.

Figure 39 shows an example.

The **peaks** are the local maxima points in the scanline. They are found by calling a *SciPy* function on the scan reflectance profile: `signalfind_peaks`. Figure 40 shows an example.

Finally, the **valleys** are the local minima points in the scanline. They are found by calling the same *SciPy* procedure of before, but applied on the function obtained by subtracting from 100 the scan reflectance profile. Figure 41 shows an example.

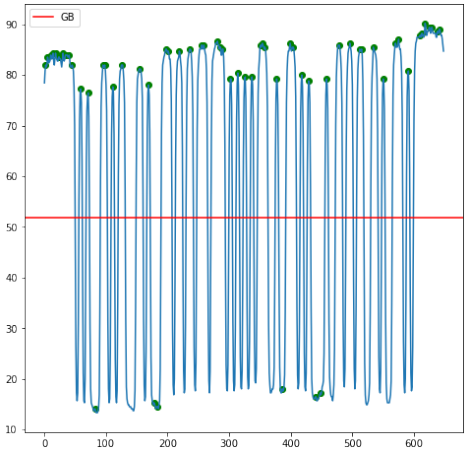


Figure 40: Peaks of the first scanline of the image 'UPC#01'

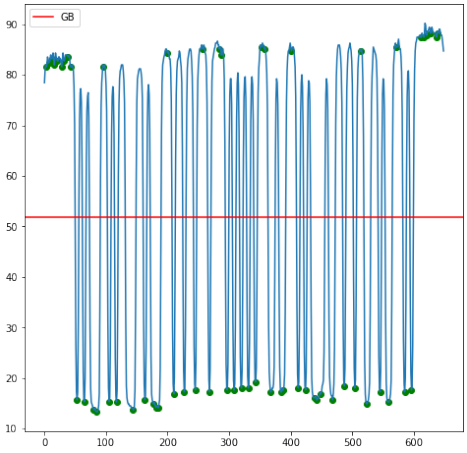


Figure 41: Valleys of the first scanline of the image 'UPC#01'

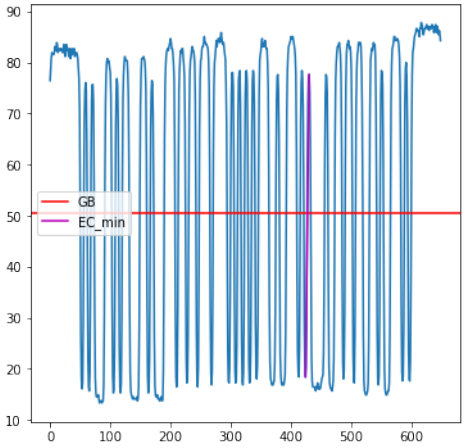


Figure 42: $EC_{\min}$ of the first scanline of the image 'UPC#01'

5.3 $EC_{\min}$ and modulation

Given an adjacent pair bar-space (or space-bar), its Edge Contrast EC is computed as the difference between the minimum peak value within the space and the maximum valley value within the bar. Intuitively, EC represents the difference between the intensities in the space and the intensities in the bar: the bigger EC, the more different these adjacent bar-space are, and, therefore, the better is the quality of that pair.
$EC_{\min}$ is the minimum EC among all possible adjacent pairs bar-space (or space-bar). The bigger is $EC_{\min}$, the better is the barcode print quality.
Figure 42 shows an example.

Then, modulation M is simply computed as $EC_{\min}/SC$.

5.4 Defect

Given a scanline element (either a bar or a space), its Element Reflectance Non-uniformity ERN is defined as the difference between the maximum peak value within that element and the minimum valley value within that element. Intuitively, ERN represents how much variability is present inside that element: the bigger ERN, the more variability is contained in that element, and, therefore, the worse is the quality of that element.
$ERN_{\max}$ is the maximum ERN among all possible scanline elements. The smaller is $ERN_{\max}$, the better is the barcode print quality.
Figure 43 shows an example.

Then, defect D is simply computed as $ERN_{\max}/SC$.

5.5 Overall symbol grade for the whole barcode

Given a certain scanline, a numerical value is computed for all the parameters $R_{\min}, SC, EC_{\min}, M, D$, as explained in the previous sections. After that, a symbolic grade between 'A' and 'F' is assigned to each parameter, by using specific rules: 'A' means very good, 'F' means very bad.
Then, a symbolic grade and a numerical value are assigned to the whole scanline.
The following table shows the computed quality parameters for the first scanline of the image 'UPC#01'.

$R_{\min}$	$R_{\min}$ grade	SC	SC grade	$EC_{\min}$	$EC_{\min}$ grade	M	M grade	D	D grade	Overall grade
13.33	A	74.50	A	59.21	A	0.79	A	0.04	A	A

Finally, an overall numerical value and an overall symbolic grade are assigned to the whole barcode, as shown in the following table for the image 'UPC#01'.

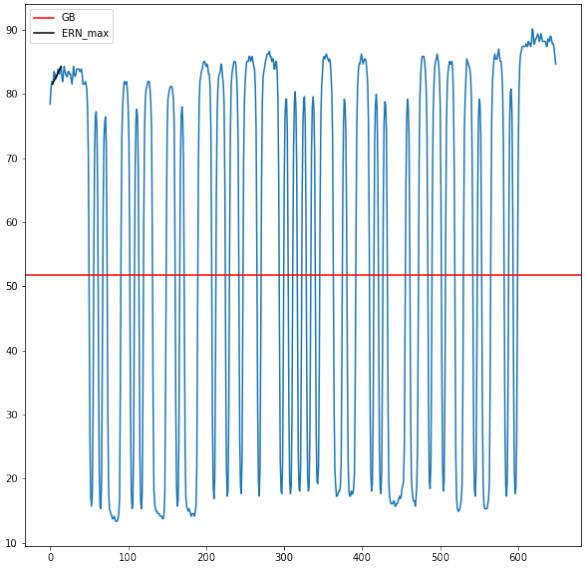


Figure 43: $ERN_{\max}$ of the first scanline of the image 'UPC#01'

Overall numerical value	Overall symbolic grade grade
4.0	A

All these computations are performed by following certain specific rules, explained in the project description file `Linear-Barcodes-Verification-Project.pdf`.

Optionally, also the overall mean quality parameters for the whole barcode are computed. For instance, also the average $R_{\min}$ value across the N scanlines and the corresponding symbolic grade are computed for the whole barcode. This is done also for SC, $EC_{\min}$, M and D. The following table shows these overall mean quality parameters for the image 'UPC#01'.

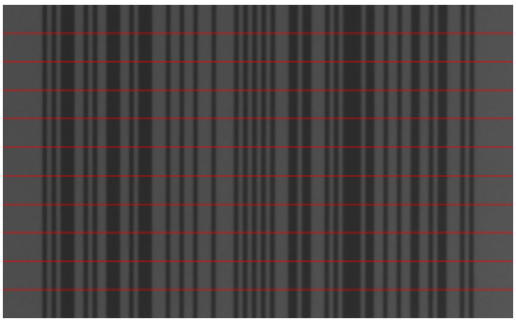
$R_{\min}$	$R_{\min}$ grade	SC	SC grade	$EC_{\min}$	$EC_{\min}$ grade	M	M grade	D	D grade
13.37	A	77.02	A	60.78	A	0.79	A	0.04	A

5.6 Accuracy of the computed quality parameters

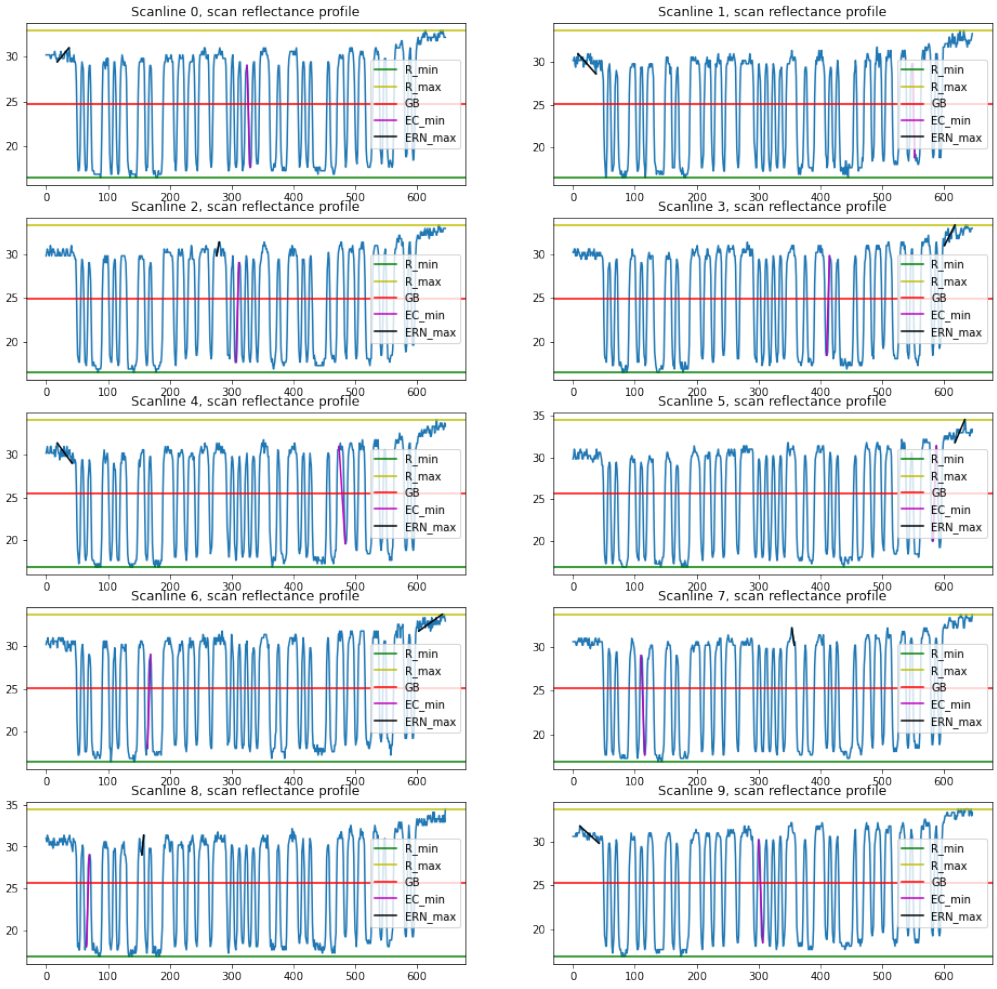
The computed quality parameters SC, M, D are pretty similar to the actual ones, which are written on the excel file contained in the dataset. This is true especially for SC. Regarding M and D, the computed values are more different with respect to the actual ones. This is normal, since M and D are more complex to compute and they require more steps, among which there is also the computation of SC itself: therefore, a small initial error can propagate to a bigger and bigger final error.

Nonetheless, the computed quality parameters are, on the whole, pretty accurate. Moreover, the results are coherent with respect to the errors and artifacts which have been added on purpose on the images. Figure 44 shows an example on a bad contrast image. Figure 45 shows an example on a bad modulation image: it can be seen that the $EC_{\min}$ is correctly detected on the modulation artifact. Figure 46 and figure 47 show two examples on bad defect images: it can be seen that the $ERN_{\max}$ is correctly detected on the defect spot and on the defect void.

However, there is one specific image for which the computed quality parameters are pretty different from the actual ones. This image, image 'I25-DEFECTS IMGB', should have a good modulation and a bad defect, due to a defect spot. Instead, the computed quality parameters reveal the opposite: a bad modulation and a good defect. This happens because the defect spot intensity is so low that, when computing the quality parameters on the scanlines, the defect spot is recognized as a bar (i.e. its value is lower than GB). Figure 48 shows this particularly bad situation.

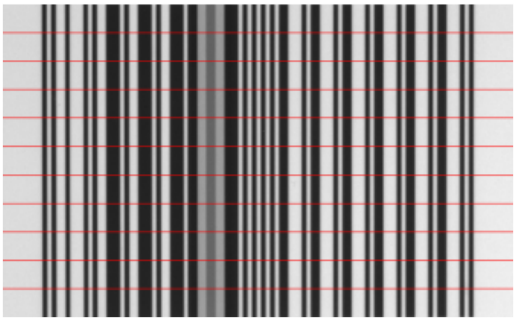


(a)

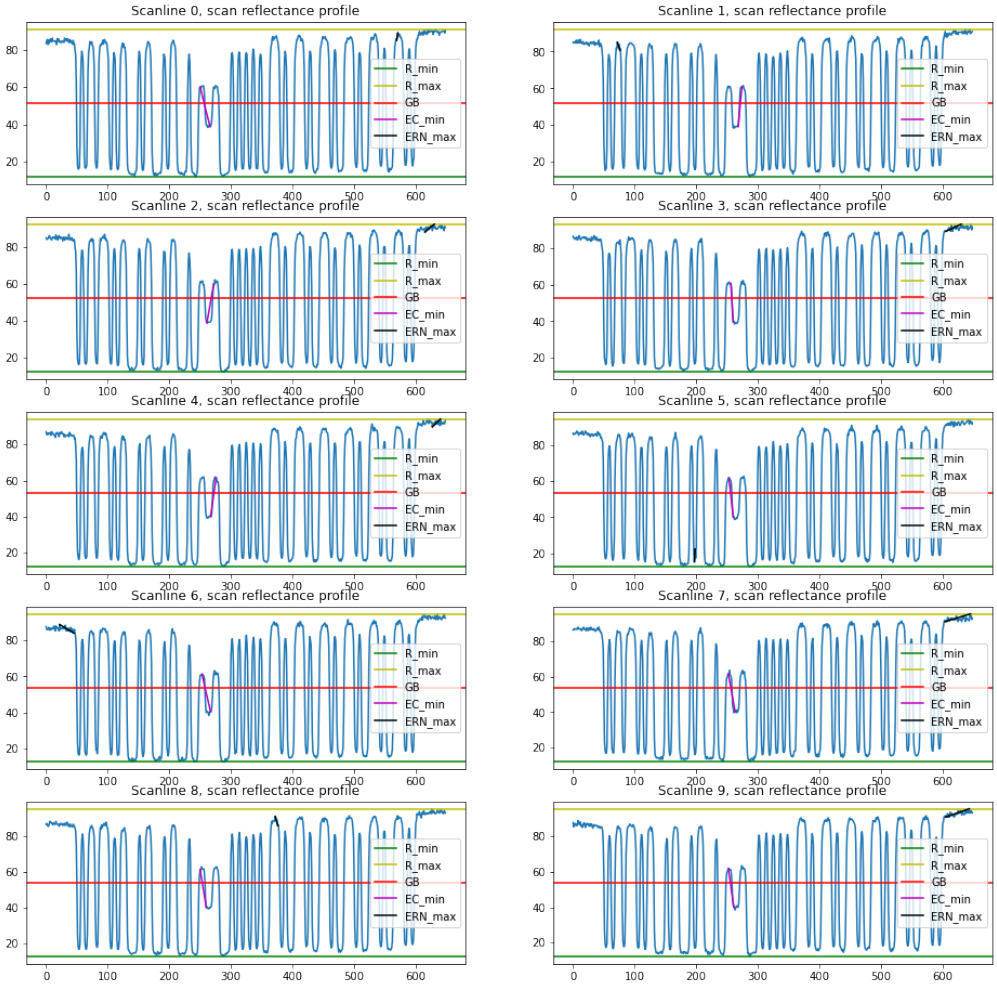


(b)

Figure 44: Bad contrast image 'UPC#07': (a) Refined ROI image, with the scanlines (b) Scanlines reflectance profile

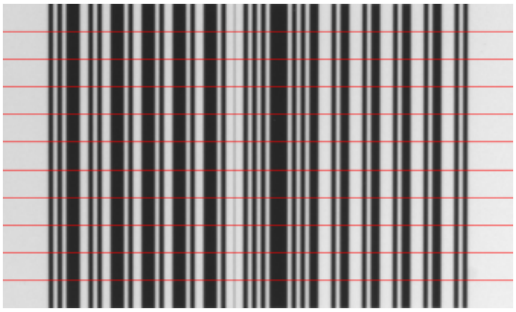


(a)

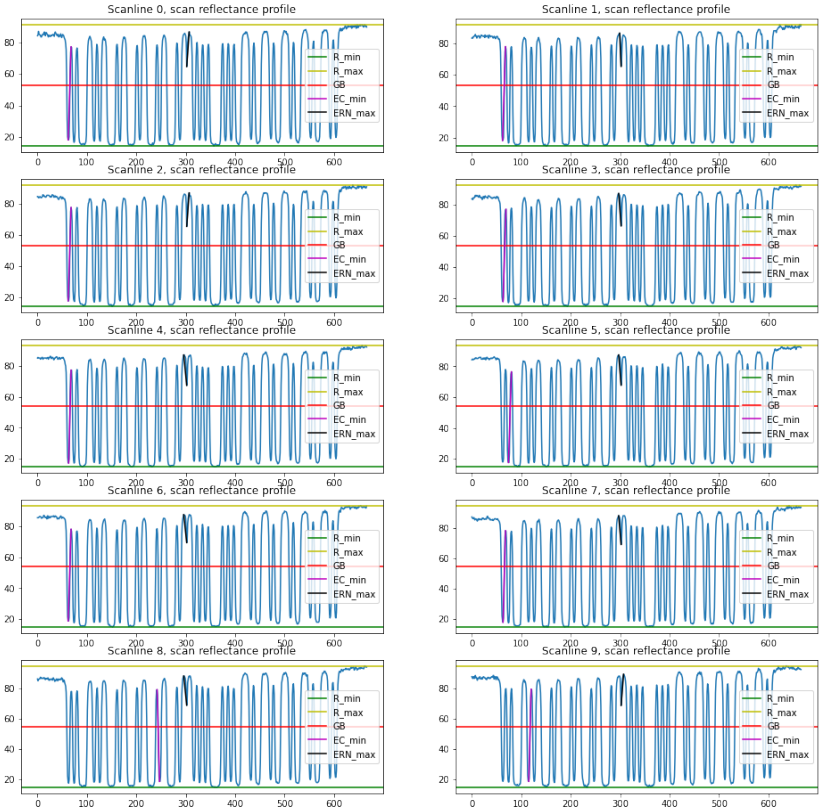


(b)

Figure 45: Bad modulation image 'UPC#12': (a) Refined ROI image, with the scanlines (b) Scanlines reflectance profile. $EC_{\min}$ is correctly detected on the modulation artifact.

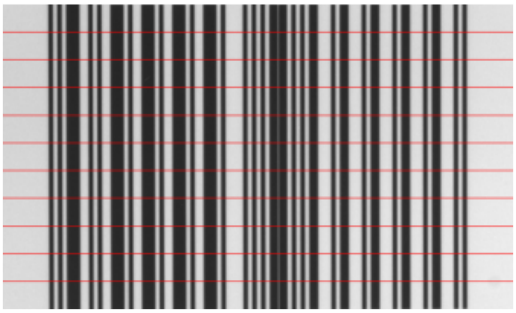


(a)

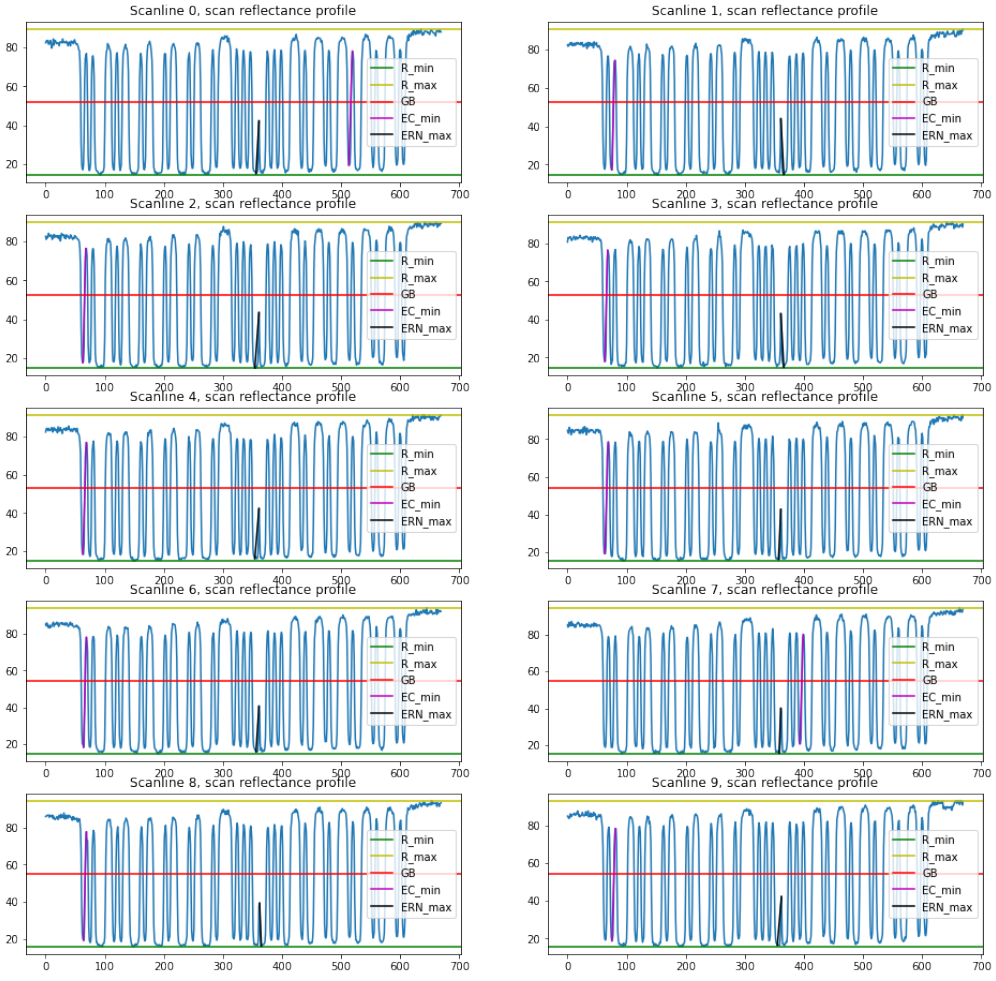


(b)

Figure 46: Defect spot image 'UPC#17': (a) Refined ROI image, with the scanlines (b) Scanlines reflectance profile. $ERN_{\max}$ is correctly detected on the defect spot.

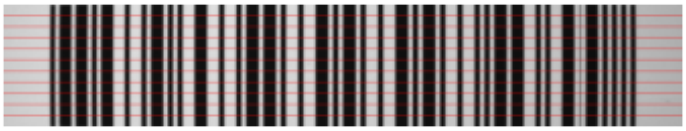


(a)

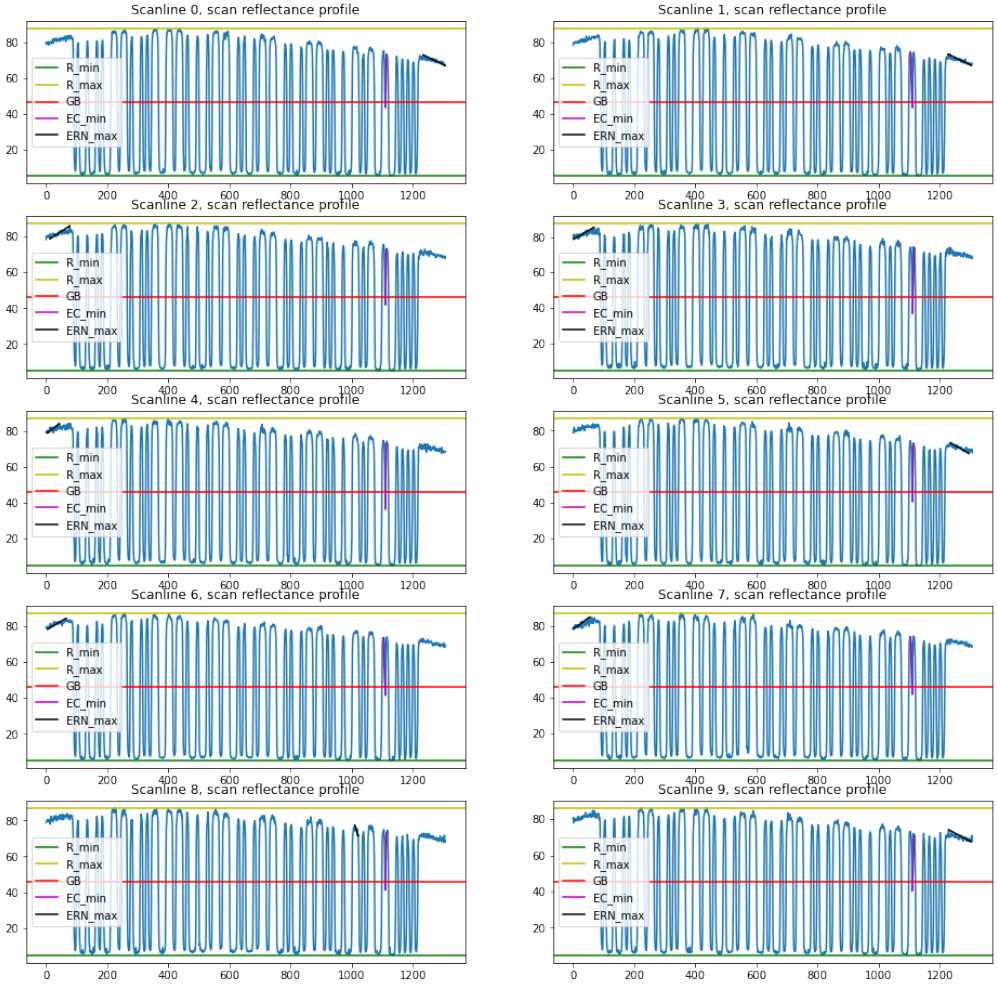


(b)

Figure 47: Defect void image 'UPC#22': (a) Refined ROI image, with the scanlines (b) Scanlines reflectance profile. $ERN_{\max}$ is correctly detected on the defect void.



(a)



(b)

Figure 48: Image 'I25-DEFECTS IMGB': (a) Refined ROI image, with the scanlines (b) Scanlines reflectance profile. This is a particularly bad scenario, because the defect spot is recognized as a bar: therefore, the computation of modulation and defect is strongly negatively affected.

6 Conslusion

In this section, some final considerations and results are shown.

First of all, the following table shows the average solving time (in seconds) of the application of the barcode print quality verification pipeline on the dataset images. The overall average solving time is 0.06 seconds.

BB detection	BB rotation	ROI refinement	Quality parameters computation
0.015	0.004	0.022	0.016

Then, the following final figures show some examples of the application of the barcode print quality verification pipeline on some dataset images.

References

[1] *Python programming language*. URL: https://www.python.org/.

[2] *NumPy library*. URL: https://numpy.org/.

[3] *SciPy library*. URL: https://scipy.org/.

[4] *OpenCV Python library*. URL: https://pypi.org/project/opencv-python/.

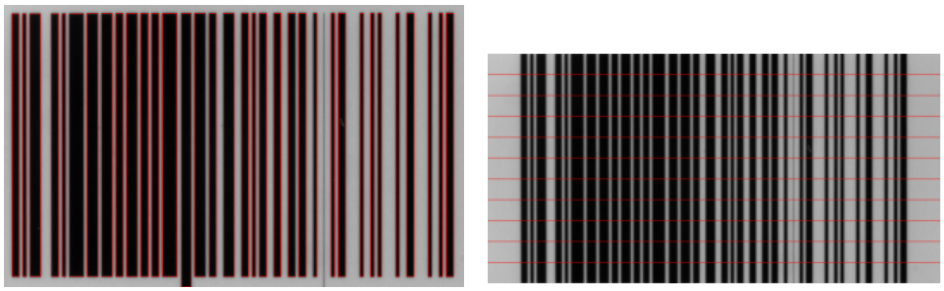
[5] *Matplotlib library*. URL: https://matplotlib.org/.

[6] *Pandas library*. URL: https://pandas.pydata.org/.

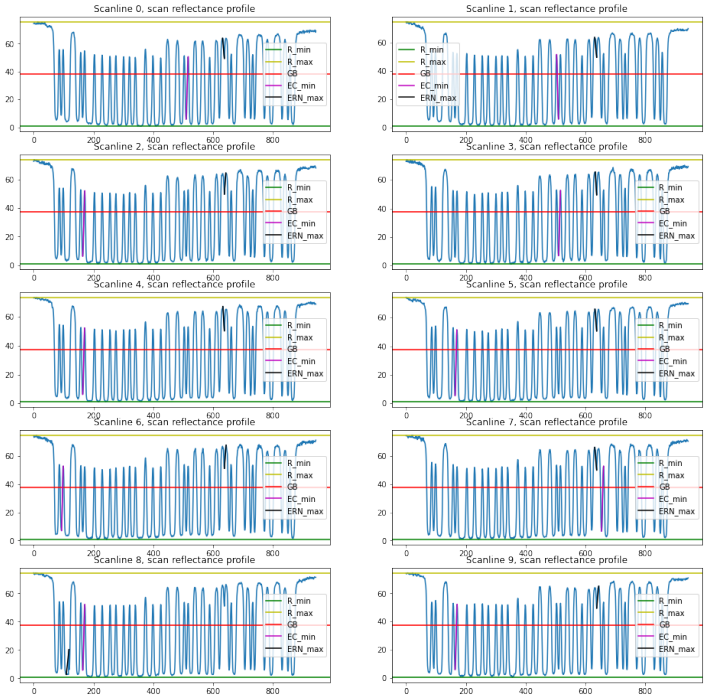
[7] *Scharr operator*. URL: https://docs.opencv.org/4.x/d5/d0f/tutorial_py_gradients.html.

[8] *Detecting outliers*. URL: https://www.khanacademy.org/math/statistics-probability/summarizing-quantitative-data/box-whisker-plots/a/identifying-outliers-iqr-rule.

(a) (b)



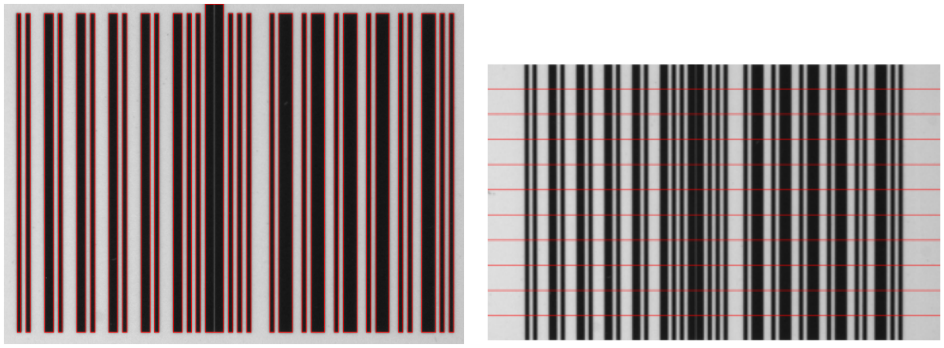
(c) (d)



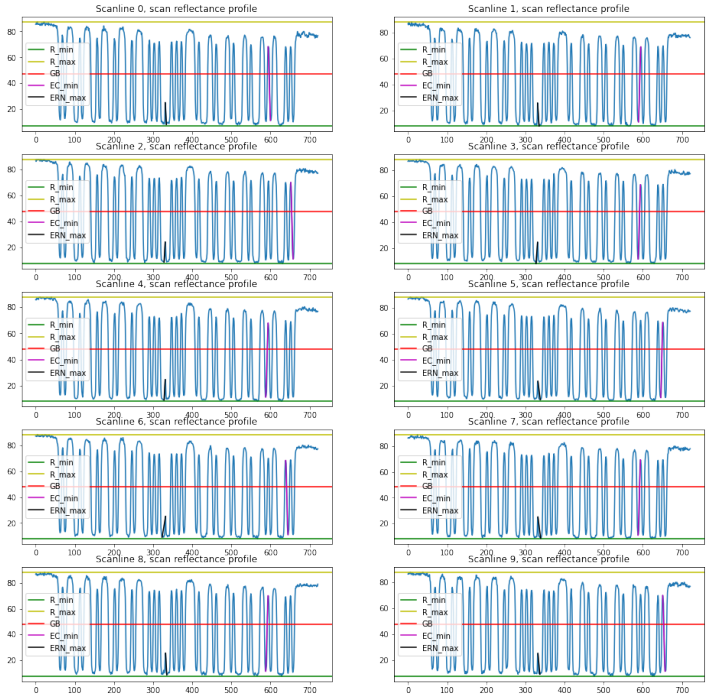
(e)

Figure 49: Image 'C128_75̇LOW': (a) Original image with the bounding box (b) Rotated image with the rotated bounding box (c) Barcode structure (d) Refined ROI image, with the scanlines (e) Scanlines reflectance profile

(a) (b)



(c) (d)



(e)

Figure 50: Image 'EAN-UPC-DEFECTS IMGB': (a) Original image with the bounding box (b) Rotated image with the rotated bounding box (c) Barcode structure (d) Refined ROI image, with the scanlines (e) Scanlines reflectance profile

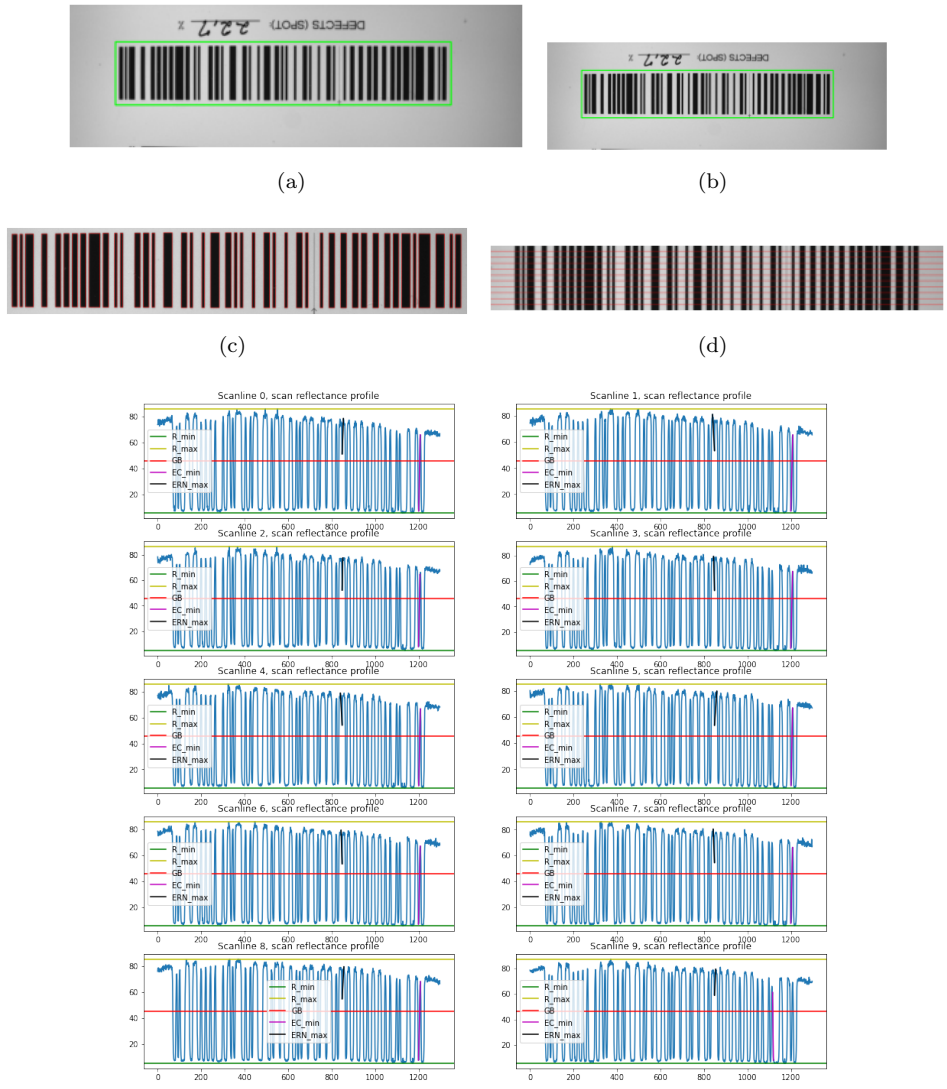
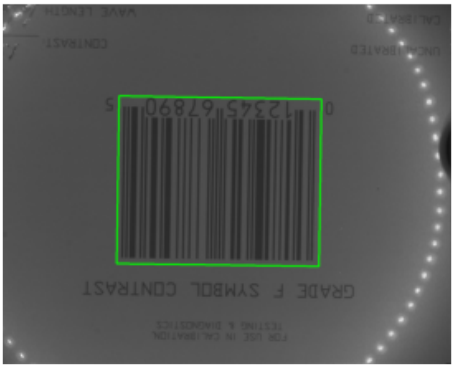
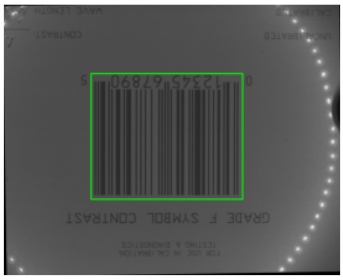


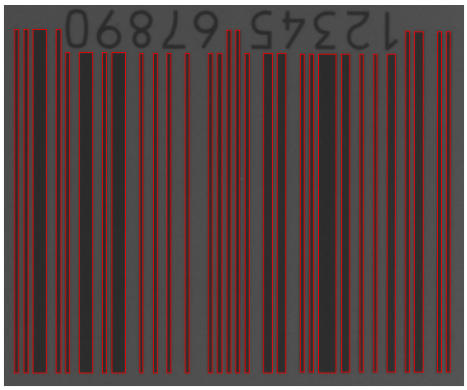
Figure 51: Image 'EAN128-DEFECTS IMGB': (a) Original image with the bounding box (b) Rotated image with the rotated bounding box (c) Barcode structure (d) Refined ROI image, with the scanlines (e) Scanlines reflectance profile



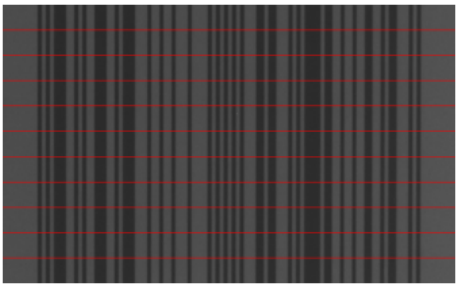
(a)



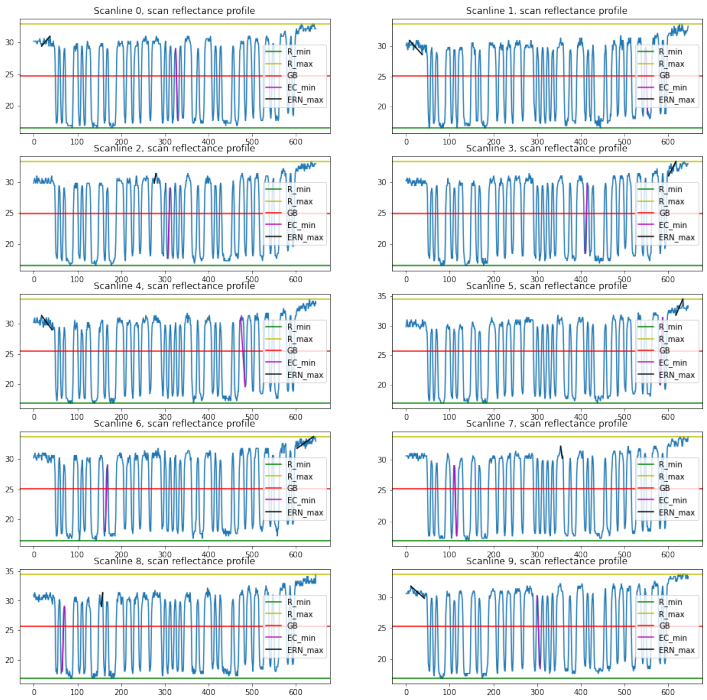
(b)



(c)



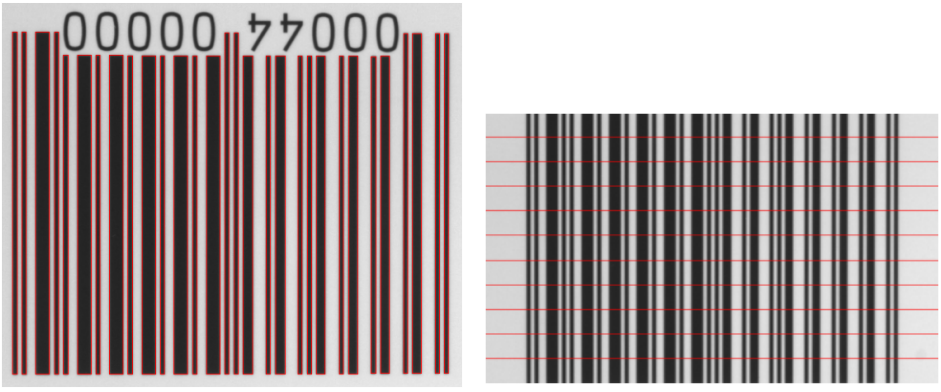
(d)



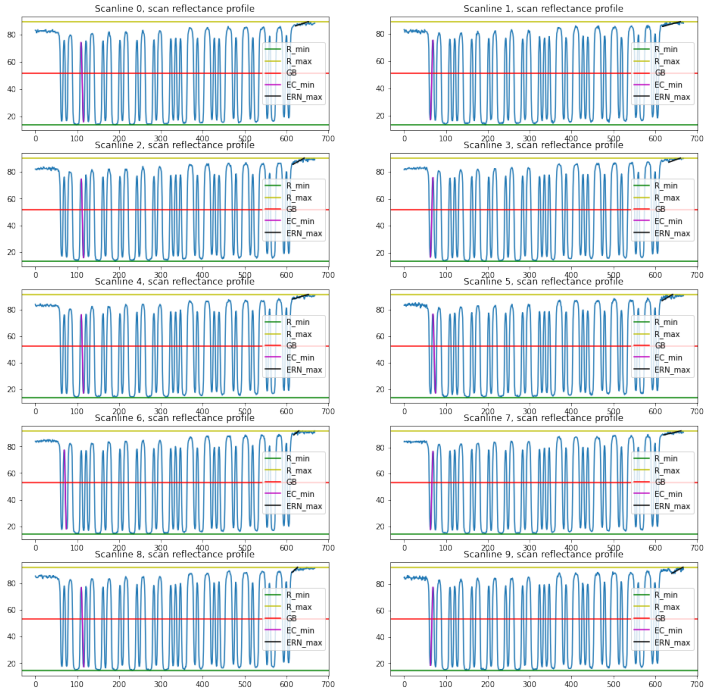
(e)

Figure 52: Image 'UPC#07': (a) Original image with the bounding box (b) Rotated image with the rotated bounding box (c) Barcode structure (d) Refined ROI image, with the scanlines (e) Scanlines reflectance profile

(a) (b)



(c) (d)



(e)

Figure 53: Image 'UPC#24': (a) Original image with the bounding box (b) Rotated image with the rotated bounding box (c) Barcode structure (d) Refined ROI image, with the scanlines (e) Scanlines reflectance profile