COMPUTER VISION REPORT

March 31, 2019

Franco Nicola
Alma Mater Studiorum
Dipartimento di Ingegneria dell'Energia Elettrica e dell'Informazione
"Guglielmo Marconi"

Contents

0.1 Target of the Project

Several images of linear barcodes are given, and it is asked to:

- Find the ROI (Region Of Interest) with the Barcode and extract some characteristics
- Estimate quality parameters of Barcode according to the specific ISO/IEC 15416

In particolar, it is requested to compute the following quality parameters:

1. Symbol Contrast
2. Reflectance
3. Min edge contrast
4. Modulation
5. Defects
6. Overall Symbol Grade

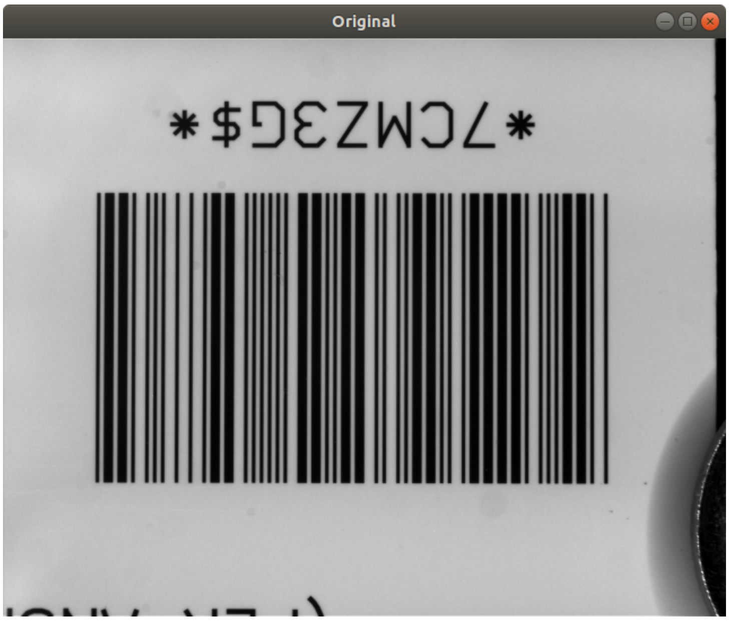

Figure 1: Example

0.2 ROI

0.2.1 Binarization

In order to produce a correct binarization of the image, I have used Otsu's Threshold, then I applied a morphology operator with a vertical line dimensional kernel, in opening configuration, to eliminate from the image different objects not relates to the barcode. After this, a new morphology operation, a dilation by a small rectangular kernel (3 rows, 1 col), several times, to connect all the regions of the barcode and creating a big one rectangle.

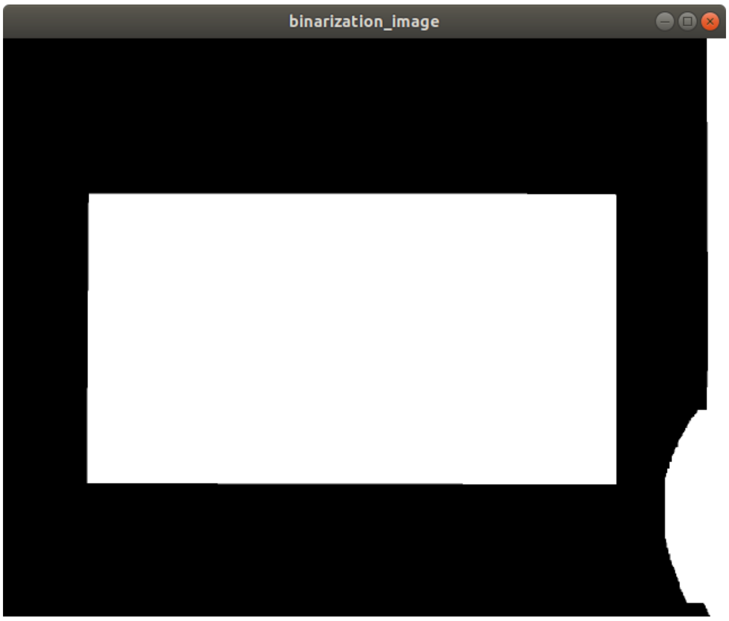

Figure 2: Binarization image

```
Mat kernel(100, 3, CV_8U, Scalar(1));
Mat element = getStructuringElement(MORPH_RECT, Size(3,1),
   Point(-1,-1));

threshold(input, binarization_image, 0, 255,
   CV_THRESH_BINARY_INV | CV_THRESH_OTSU);
morphologyEx(binarization_image, output, MORPH_OPEN, kernel);
dilate(binarization_image, binarization_image, element, Point
   (-1,-1), 15);
```

0.2.2 Labeling

According to the Flood-fill approach, I can find all the labels. Furthermore, to find exactly my barcode I compute all the areas and I estimate the biggest one. in this manner the biggest has to be the area of the barcode.

```
for (int i = 0; i < input.rows; i++)
      for ( int j = 0; j < input.cols; j++)
            areas[input.at<uchar>(i,j)]++;

for (unsigned int i = 1; i <= labels; i++)
      if(areas[i] > barcodeArea){
            barcode = i;
            barcodeArea = areas[i];
      }
```

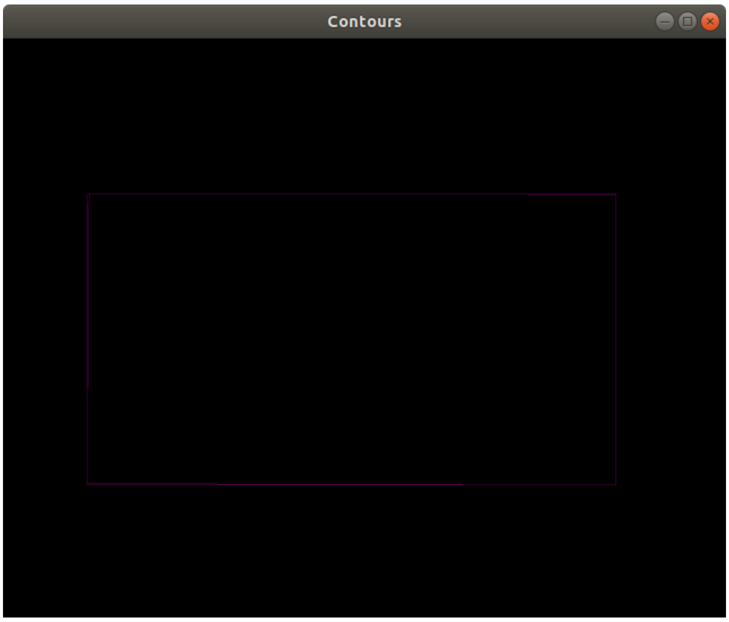

Figure 3: Labeling image

Then I want to eliminate all the other labels, with areas less than my barcode

```
for ( int i = 0; i < input.rows; i++)
         for ( int j = 0; j < input.cols; j++)
               if(input.at<uchar>(i,j) == barcode)
```

```
                output.at<uchar>(i,j) = 255;
            else
                output.at<uchar>(i,j) = 0;
```

0.2.3 Position and Orientation

In this section, I have used the function RotatedRect that it gives me back the minimum enclosing rectangle(MER). And In order to find the minimum and maximum x and y position of the barcode, I have checked all the corners of my MER.
While, to find the angle, I have used the difference between two y positons of respectively, the right and left corner along the x direction, i.e. the same side of the rectangle.

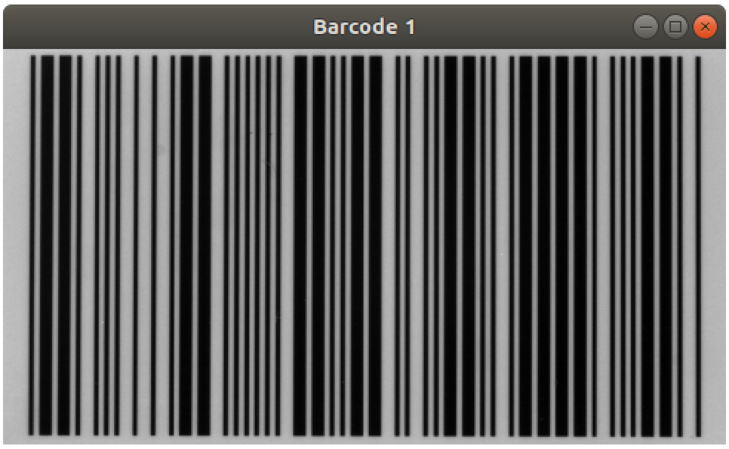

Figure 4: ROI image

```

for(int i = 0; i < 4; i++)
{
    if(rect_points[i].x < x0)
        x0 = rect_points[i].x;
    if(rect_points[i].x > x1)
        x1 = rect_points[i].x;
    if(rect_points[i].y < y0)
        y0 = rect_points[i].y;
    if(rect_points[i].y > y1)
        y1 = rect_points[i].y;
```

```

   int diff_y = rect_points[i+1].y - rect_points[i].y;
   if((abs(diff_y) > abs(angle)) && (abs(diff_y)<20))
      angle = diff_y;
}
```

At the end, by using warpAffine function, I create a rotation matrix with the angle founded before, and I rotated and cutted the image.

```
if(angle != 0){
      Mat rotation = getRotationMatrix2D(Point2f(barcode_image.cols/2,barcode_image.rows/2), angle/10, 1);
      warpAffine(barcode_image,barcode_image, rotation, Size(barcode_image.cols,barcode_image.rows));
}

output = barcode_image(Range(10,barcode_image.rows -10),Range(10,barcode_image.cols -10));
```

0.2.4 X-dimension

To get a precise estimation of my barcode, I just cutted my original image around the region of my barcode, with the previous steps. Then, I will apply again the binarization with the Otsu's threshold and the morphology open operation with a vertical line element. Now, I create for each bar a bounded rect that contains the bar, after the estimation of the contours.

```
vector<vector<Point>> contours;
vector<Vec4i> hierarchy;

// Find contours
findContours(input, contours, hierarchy, CV_RETR_CCOMP, CV_CHAIN_APPROX_SIMPLE, Point(0,0) );

```

```
// Find rectangle
vector<Rect> boundRect(contours.size());
```

Then to find the X dimensions requested of smallest bar of barcode, I evaluated all the bars in a for loop and I extracted the height, width and area.

```
for(unsigned int i = 0; i< contours.size(); i++ )
{
   //Create bounded rectangles for every bars in barcode
   boundRect.at(i) = boundingRect(contours.at(i));

   //Variables defined in order to compare with the desired
```

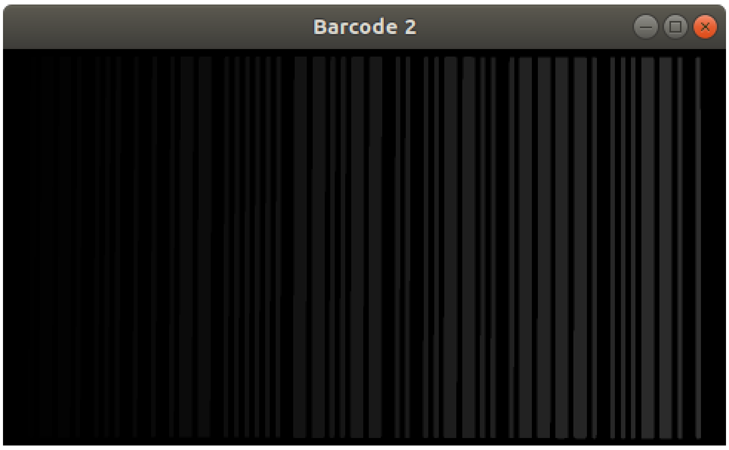

Figure 5: Contours image

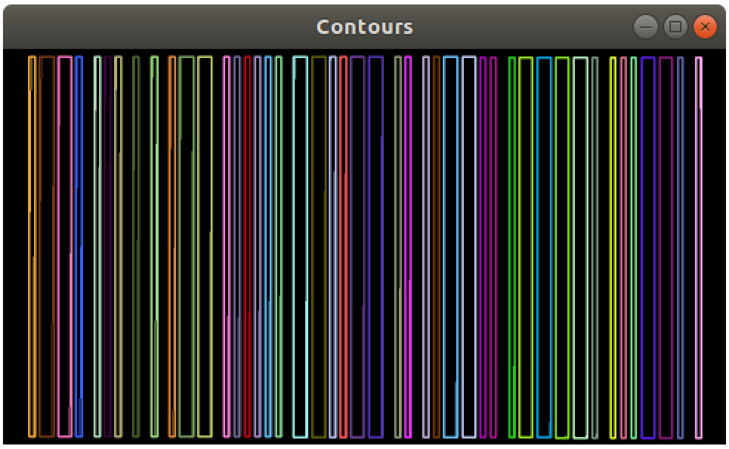

Figure 6: X-ROI image

```
    ones
   int height = boundRect.at(i).height;
   int width = boundRect.at(i).width;
   int area = height*width;

   //Find the y-length
   if( height < heightX && height > input.rows/2)
   {
      heightX = height;
      smallBarPosition[0] = boundRect.at(i).y;
      smallBarPosition[1] = boundRect.at(i).y + heightX;
   }
   //Find the area
   if( area < smallBarArea)
      smallBarArea = area;
   //Find the X_DIMENSION
   if( width < widthX)
      widthX = width;

   //Find the position of barcode in x-axis
   if(boundRect.at(i).x < smallBarPosition[2])
      smallBarPosition[2] = boundRect.at(i).x;
   if(boundRect.at(i).x > smallBarPosition[3])
   smallBarPosition[3] = boundRect.at(i).x + width;

}
```

At the end, I store all the parameters in my parameter's vector and I cut again the image in the specific ROI.

```
//Outputs of parameters
   parameters[DIMENSION_X][0] = widthX;
   parameters[HEIGHT][0] = heightX;
   parameters[POS_Y1][0] = smallBarPosition[0];
   parameters[POS_Y2][0] = smallBarPosition[1];
```

```
    parameters[POS_X1][0] = smallBarPosition[2];
    parameters[POS_X2][0] = smallBarPosition[3];

//Cut the barcode
    smallBarPosition[2] = smallBarPosition[2] - widthX*10;
    smallBarPosition[3] = smallBarPosition[3] + widthX*10;

output = source(Range(smallBarPosition[0],smallBarPosition[1])
    ,Range(smallBarPosition[2],smallBarPosition[3]));
```

0.3 Barcode Parameters

I started to create 10 parallel lines, by dividing the number of rows of my ROI, then I stored the pixels values for each line into an scan_profile vector, with a for loop.
I founded the first two parameters: Minimum Reflectance and Symbol Contrast.

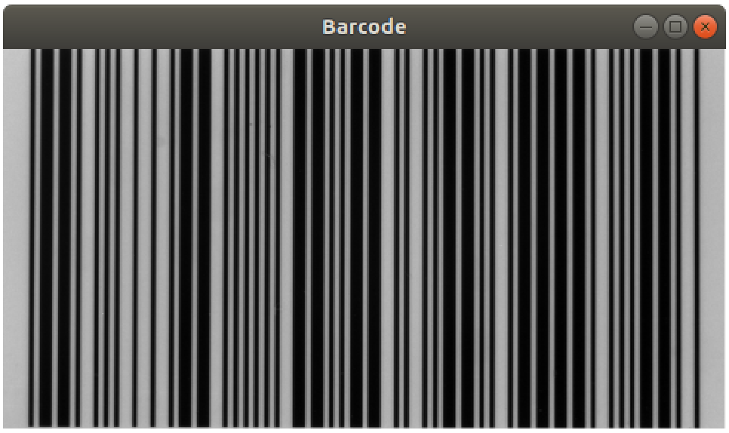

Figure 7: X-dimensional ROI Barcode

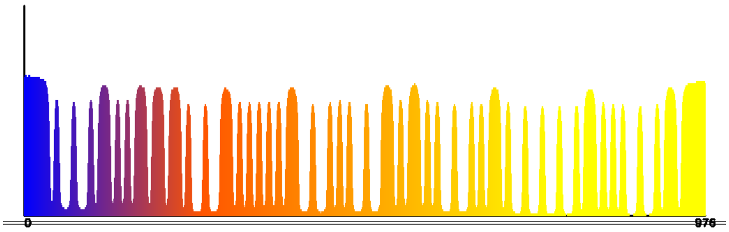

Figure 8: Scan profile

```
for(int j = 0; j < input.cols; j++)
{
   //I am scanning each pixel in the row and then compute the
   percentage
   scan_profile[i][j] = (int) input.at<uchar>(row,j)*CONV;

   if(scan_profile[i][j] < parameters[Rmin][i])
      parameters[Rmin][i] = scan_profile[i][j];
   if(scan_profile[i][j] > parameters[Rmax][i])
      parameters[Rmax][i] = scan_profile[i][j];
}

parameters[SC][i] = parameters[Rmax][i] - parameters[Rmin][i];
```

Then in another loop, I have defined the median line equal than the half of the Symbol Contrast SC/2. After that, I compute the frei chen operator, to evaluate the forward and backward pixels along the scan, in order to reduce the noise. So, when I am in the proximity of my median line, with two thresholds, one for the upper part and the other for the lower part of the edge, I can compute the edge contrast and update the number of edges founded.

```
//When I match the mean, there are 2 situations: ascendent or
   descendet edge:
foreground = (scan_profile[i][j] + pow(2,0.5)*scan_profile[i][
   j+1] + scan_profile[i][j+2])/(2+pow(2,0.5));
background = (scan_profile[i][j-2] + pow(2,0.5)*scan_profile[i
   ][j-1] + scan_profile[i][j])/(2+pow(2,0.5));

   //The ascendent
   if(background < median && foreground > median &&
   scan_profile[i][j-1] < scan_profile[i][j+1])
   {
      parameters[EDGES][i]++;
      flag = true;
      count1 = j;
      count2 = j;

      //Compute the upper part
      do{
         edge_up = scan_profile[i][count1];
         count1++;
      }while(scan_profile[i][count1-1] < scan_profile[i][
   count1]);

      //Compute the down part
      do{
         edge_down = scan_profile[i][count2];
         count2--;
      }while(scan_profile[i][count2] < scan_profile[i][count2
   +1]);
```

A similar procedure I have adopted to find the defects:

```
//Compute Defects
count_ern = count1;
ern_down = 100;
ern_up = 0;

   //Compute until we don't overtake the threshold
   do{
   if(scan_profile[i][count_ern] > ern_up)
      ern_up = scan_profile[i][count_ern];

   //Update until we don't find the defect
   while(scan_profile[i][count_ern] > scan_profile[i][
   count_ern+1])
      count_ern++;

   if(scan_profile[i][count_ern] > ern_down && scan_profile[i
    ][count_ern] > median)
        ern_down = scan_profile[i][count_ern];

        count_ern++;
    }while(scan_profile[i][count_ern] > median);
```

Equal format it is used for the downhill edge. At the end, If I find an edge I can compare the parameters with the desired one and I will store the parameters into the parameter's vector.

```
if(flag)
{
    //I compute the edge size and then I store in parameters
    vector
    edge = edge_up - edge_down;
    if(edge < parameters[ECmin][i] && edge > N)
        parameters[ECmin][i] = edge;

    // I store the defect value
    ern = ern_up - ern_down;
    if(ern > parameters[ERN][i] && ern < N*5)
        parameters[ERN][i] = ern;
    }

    flag = false;
}

parameters[MOD][i] = (parameters[ECmin][i]/parameters[SC][i])
    *100;
parameters[Defects][i] = (parameters[ERN][i]/parameters[SC][i
    ])*100;
```

0.3.1 Overall Symbol Grade

To find the Overall Symbol Grade, I have created a comparision between all the parameters, for all the parallel scans. Then I compute the mean value between them.

```
//Variables
int Symbol, Overall = 0;

for(int i = 0; i < N; i++)
{
    Symbol = 4;

    //Rmin
    if(parameters[Rmin][i] <= parameters[Rmax][i]/2)
        Symbol = 4;
    else
        Symbol = 0;

    //ECmin
    if(parameters[ECmin][i] >= ECmin_GRADE && Symbol > 3)
        Symbol = 4;
    else
        Symbol = 0;

    //Symbol Contrast
    if(parameters[SC][i] >= SC_GRADE_A && Symbol > 3 )
        Symbol = 4;
    else if (parameters[SC][i] >= SC_GRADE_B && parameters[SC][
    i] < SC_GRADE_A && Symbol > 2)
        Symbol = 3;
    else if (parameters[SC][i] >= SC_GRADE_C && parameters[SC][
    i] < SC_GRADE_B && Symbol > 1)
        Symbol = 2;
    else if (parameters[SC][i] >= SC_GRADE_D && parameters[SC][
    i] < SC_GRADE_C && Symbol != 0)
        Symbol = 1;
```

```
 else if (parameters[SC][i] < SC_GRADE_D)
     Symbol = 0;

 //Modulation
 if(parameters[MOD][i] >= MOD_GRADE_A && Symbol > 3)
     Symbol = 4;
 else if (parameters[MOD][i] >= MOD_GRADE_B && parameters[
MOD][i] < MOD_GRADE_A && Symbol > 2)
     Symbol = 3;
 else if (parameters[MOD][i] >= MOD_GRADE_C && parameters[
MOD][i] < MOD_GRADE_B && Symbol > 1)
     Symbol = 2;
 else if (parameters[MOD][i] >= MOD_GRADE_D && parameters[
MOD][i] < MOD_GRADE_C && Symbol != 0)
     Symbol = 1;
 else if (parameters[MOD][i] < MOD_GRADE_D)
     Symbol = 0;

 //Defects
 if(parameters[Defects][i] <= Defects_GRADE_A && Symbol > 3)
     Symbol = 4;
 else if (parameters[Defects][i] <= Defects_GRADE_B &&
parameters[Defects][i] > Defects_GRADE_A && Symbol > 2)
     Symbol = 3;
 else if (parameters[Defects][i] <= Defects_GRADE_C &&
parameters[Defects][i] > Defects_GRADE_B && Symbol > 1)
     Symbol = 2;
 else if (parameters[Defects][i] <= Defects_GRADE_D &&
parameters[Defects][i] > Defects_GRADE_C && Symbol > 0)
     Symbol = 1;
 else if (parameters[Defects][i] >= Defects_GRADE_D )
     Symbol = 0;

 parameters[SYMBOL][i] = Symbol;
 Overall = Overall + Symbol;
```

```
}
parameters[OVERALL][0] = Overall/N;
```

In conclusion, I created a function to print all the parameters founded previously in the barcode, to a excel file.