BARCODE DETECTION

Nguyen Minh Anh Hao

May 21 2021

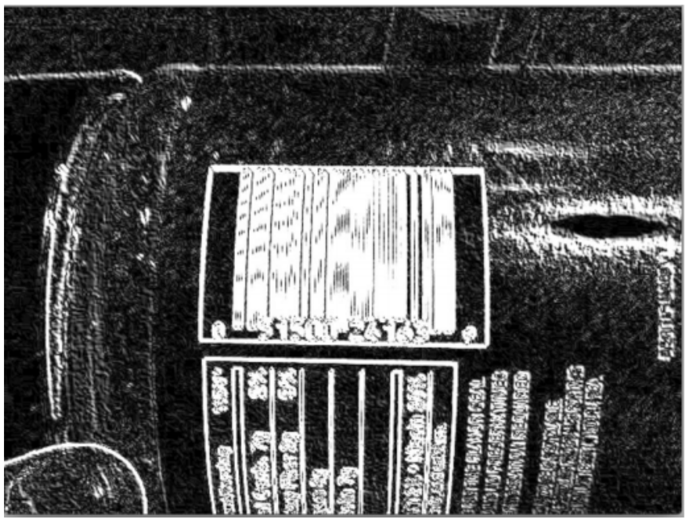
1 Introduction

As consumers, barcodes and barcode scanners are relatively familiar concept: purchasing from any retail store, renting a car, attending major events, flying, and even going to the dotor. They're in our social media apps and on store windows.
The goal of this algorithm post is to demonstrate a basic implementation of barcode detection using computer vision and image processing techniques.

2 The Algorithm

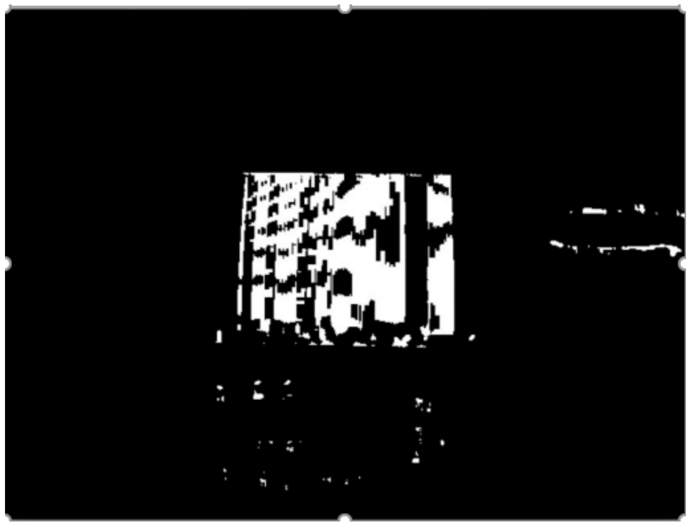
1. **Detecting Barcodes In Images**
   The first thing we'll do is import the packages:

   - NumPy for numeric processing.argparse for parsing command line arguments, and cv2 for our OpenCV.
   - Then we'll setup our command line arguments. We need just a single switch here, –image , which is the path to our image that contains a barcode that we want to detect.bindings.
   - Then, we use the Scharr operator (specified using ksize = -1) to construct the gradient magnitude representation of the grayscale image in the horizontal and vertical directions.
   - From there, we subtract the y-gradient of the Scharr operator from the x-gradient of the Scharr operator.The result after processing:

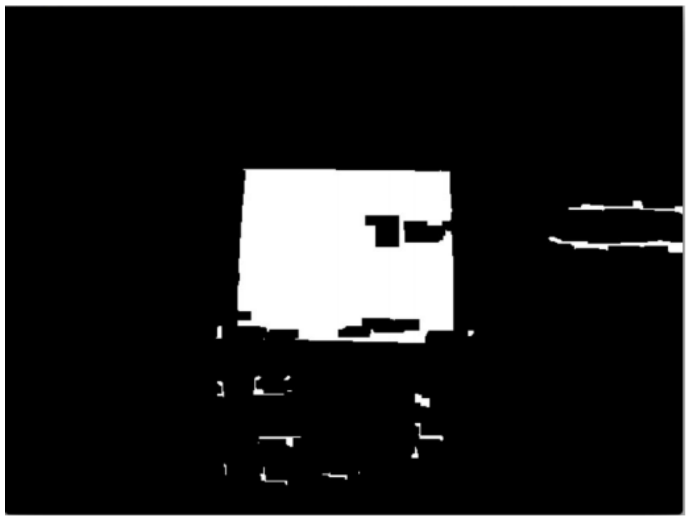
   

   The next steps will be to filter out the noise in the image and focus solely on the barcode region (apply an average blur to the gradient image using a 9 x 9 kernel, then threshold the blurred image. Any pixel in the gradient image that is not greater than 225 is set to 0 (black). Otherwise, the pixel is set to 255 (white)).

   - The output of the blurring and thresholding looks like this:



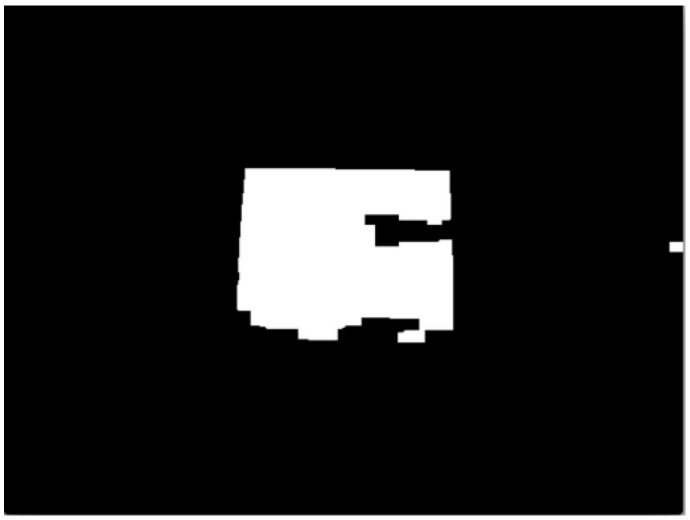
- We'll start by constructing a rectangular kernel using the cv2.getStructuringElement
- This kernel has a width that is larger than the height, thus allowing us to close the gaps between vertical stripes of the barcode.
- The gaps are substantially more closed, as compared to the thresholded image above:



- All we are doing here is performing 4 iterations of erosions, followed by 4 iterations of dilations.
- An erosion will "erode" the white pixels in the image, thus removing the small blobs, whereas a dilation will "dilate" the remaining

white pixels and grow the white regions back out.

- Provided that the small blobs were removed during the erosion, they will not reappear during the dilation.

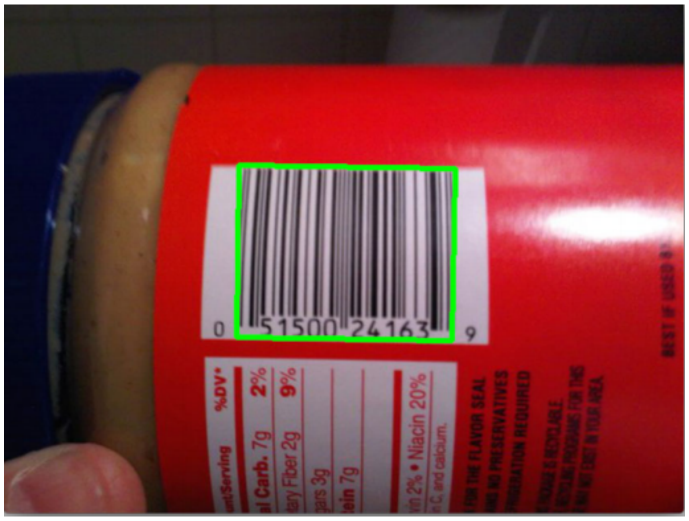


Finally, find the contours of the barcoded region of the image:

- We simply find the largest contour in the image, which if we have done our image processing steps correctly, should correspond to the barcoded region.

- We then determine the minimum bounding box for the largest contour and finally display the detected barcode.

- Here is the final result:



2. **Barcode Detection In Frames Of A Video**
   The first thing we'll do is import the packages we'll need — NumPy for numeric processing and cv2 for our OpenCV bindings.

   - From there define our detect function. This function takes a single argument, the image (or frame of a video) that want to detect a barcode in. Converts our image to grayscale and find regions of the image that have high horizontal gradients and low vertical gradients.
   - We then blur and threshold the image so we can apply morphological operations to the image. These morphological operations are used to reveal the rectangular region of the barcode and ignore the rest of the contents of the image.
   - If no outline can be found, then we make the assumption that there is no barcode in the image.
   - However, if we do find contours in the image, then we sort the contours by their area

   Finally, we take the contour and compute its bounding box. This will give us the (x,y) coordinates of the barcoded region, which is returned to the calling function

3 Benchmarks

For the benchmarks, we are going extend barcode detection algorithm and refactor the code to detect barcodes in video.
We are comparing an algorithm speed for an algorithm without any parallelization with the parallelized algorithm.
For the speed of the algorithm, I am measuring two different times.
CPU Time Sum of time on all threads in ms.
Real Time A real time from the beginning of the algorithm until the algorithm is finished in ms.

We are comparing detect video without parallelization with detect video with parallelization
No Parallelization :

CPU Time 22.497857332229614 [s]   Real Time [s] 22.507866859436035

Parallelization (5 processors) :

CPU Time 8.984466552734375 [s]   Real Time 8.995312690734863 [s]

4 Conclusion

Although the duration of the execution is considerably shorter, the increase in difficulty of the task is significant.Either the parallelization needs to be better optimised this algorithm will not work for all barcodes, but it should give you the basic intuition as to what types of techniques you should be applying. I implemented the algorithm to read both barcodes and QR codes but only for the basic code. With some complex code, my algorithm will not work efficiently enough.