Final Project - Final Report

SCC 0251 - Image Processing

GROUP 4
Kelvin Guilherme de Oliveira - 9293286
Mauricio Caetano da Silva - 9040996

1. MAIN OBJECTIVE

This project aims the development of a software be able to perform the price recognition of a product in a department store, from a photography taken by a cell phone.

The main function of this software is to facilitate the development of other applications that needs this service, for example, an application that performs the purchase total value of a client in a supermarket from photos of the product prices, either an application that tells to the client whether the product to which the photo was taken is expensive or not, among other applications.

2. INPUT IMAGES

The images used as input to our software are images photographed of the price tags related to products sold in a department store.

We insert some restrictions on the images in order to facilitate the software development. Are the following:

- The image must contain just one price tag relative to a product. Images with more than one price tag will not return valid results.
- The price tag should contain the rectangular shape, and it should not be overlapping by other objects or price tags on the photo.
- The price tag should be green or yellow. Any price tags in different colors will not be found.
- The photo should be taken with the cellphone in a paralell plan of the price tag. In other words, the price tag cannot show a perspective idea (More details on section 3.2.4).

In order to perform tests and to improve our software, we used photos taken of some supermarket shelves, placed on Sao Carlos (Sao Paulo, Brazil), using the cameras of the cell phones Asus Zenfone 5 e Motorola Moto G2.

The images were taken in different times with different lighting conditions and angulation in several supermarkets, with the goal in purpose a better adaptation in variant environments where the software will be subject. Following there are some examples of captured images.

3. DESCRIPTION OF STEPS AND METHODS USED

This project has two parts defined very well and independent. Because that, we will explain the performed steps and to explore the results obtained separately, in order to after, on section 4 (Final Results), we show the parts union and the final solution of this project. The two parts are:

- Price tag extraction from the original image.
- Price extraction from a price tag.

3.1. PRICE TAG EXTRACTION

The first part of this project was aimed at locating the price tag on a photo, so that we can later extract its numerical value using the Tesseract recognition software.

The price label extraction flow is formed by the following steps: Pre-process the photo of the shelf with products, highlighting the contours of the label and excluding the edges outlines of the other objects to the maximum, Extract the objects that with greater probability of to be the price tag, Extract the label by copying its contents to a second image that will be used later. Later will be explained step by step, the ideas and methods used, as well as the problems encountered. The process flow is shown in the image below.

3.1.1. Shelf image preprocessing

To highlight the contours of the price tags the first step was the conversion of the color system. By default, reading an image in the OpenCv library is performed on the RGB system, so we convert your RGB system to HSV. This conversion is performed in order to facilitate the differentiation of the price tag from the rest of the image by means of its color intensity (green and yellow).

After that, a binary (black and white) mask of objects with the colors involved is generated using minimum and maximum values of intensities. We then use another OpenCv function called "canny" for the extraction of edges resulting in an image of contour lines. This image still underwent some morphological operations in order to improve the lines and reduce the noise.

3.1.2 Price tag extraction

Having the image with the edges of the objects, a function called "findContours" was used that the contours of the objects were highlighted. These contours were stored in the memory, sorted by area and then we extracted a sample with the 200 largest contours.

With this sample of contours, we use another function called "approxPolyDP" to perform the approximation of these contours by polygons that fill it with the smallest possible area. Thus we were able to eliminate polygons that have more than 4 sides and that are not within a range of values of predetermined area determined by us. The range of values was defined through the individual analysis of the labels in each image.

At this point the vast majority of false positives have already been eliminated and, if the tag was found, it will be displayed on the screen.

3.1.3. Price Label Extraction Issues

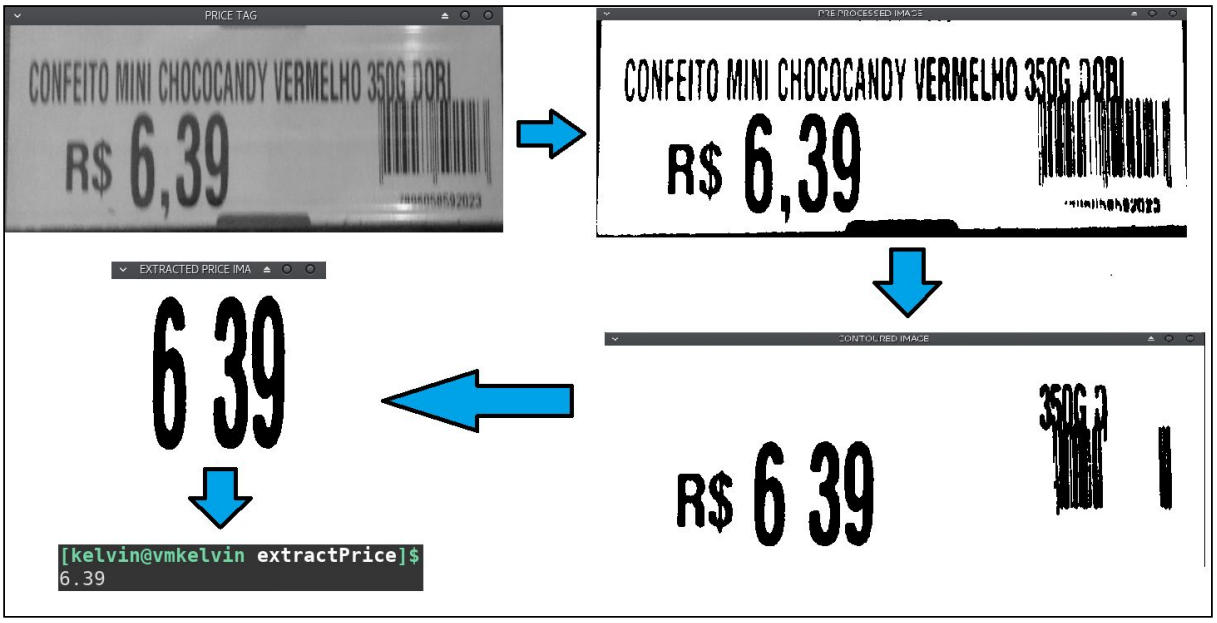
Our software did not have such high accuracy in extracting the price tag due mainly to problems in extracting the edges of the label because they are the edges of the image, the main way to find the label.

The color approach in the HSV system brought a good improvement to accuracy, but errors occur when targets that are just below the lower bound of the label have colors close to the green and yellow, so that the lines of this object are confused with the tag.

3.2. PRICE EXTRACTION

The second part of this project consists in the price extraction, in other words, to return a String containing the product value from a price tag image. This part is related with to take the extracted price tags (explained in the above section), to apply some image processing methods in order to improve the price visibility to facilitate the Tesseract (OCR, will be explain forward) recognition.

The price extraction flow is composed by: Pre-process the price tag image in order to become price more visible, to extract the objects more likely to belong the price, to extract the objects that really composes the price, to send an image to Tesseract OCR in order to receive the desired String. Following will be explained the step-by-step, the ideas and methods used, beyond the problems found in each step. The process flow is showed below.



3.2.1. Price tag image pre-processing

The first stage consisted in to prepare the price tag image in the best possible way to facilitate the price extraction. To do that, the first action was to convert the image to grayscale (to apply the desired methods) and to resize all images to 800x400, because the price tags normally have a standard size, and that facilitate the measurement calculations, that will be explained forward.

In order to highlight the price and to remove noises, some methods were tested and they have not the desired result, such as: Enhancement filters (Gaussian Blur, Logaritmic), Adaptative Filter, among others. But the best method that caused the desired response was the Threshold. In this method, we have different approaches that take to different results. It was tryed the THRESH_BINARY and the ADAPTIVE_THRESHOLDING, both without success. The best found approach was the Threshold with THRESH_BINARY together with THRESH_OTSU (parameters threshold = 127, max value = 255, generating a binary image), that results is showed below. Note that the approach highlights very well the price inthin the price tag.

3.2.2. Extraction of the most probable objects

This process stage aims to extract the image components that have more chances of belong the price, in other words, to remove the components that certainly are not related to the price contained in the price tag.

In order to realize this goal, it was used a fundamental function in our project. The function “findContours” of the OpenCV library. This function finds the contours in a binary image, returning theirs coordinates. This fact was extremely useful to we recognize the components shape and theirs properties, like area, perimeter, among others.

With such data, we could extract the components more similar to the price, discarding the rest. This selection was done based on component area (because with the contours, we can find theirs internal areas with the function “contourArea” of OpenCV Library). Analyzing the input images described on section 2, we could note that the area of the numbers that compose the price are around 3000 to 7000 (we insert a big margin of error to avoid that numbers be discarded in this stage), being ‘1’ the numeral with less area, and some numbers like ‘8’ and ‘9’ being the large numbers. Therefore, we execute a Threshold in the components, leaving just those that have the area between the values that characterize the numbers.

So the image composed by the possible elements stays like the following images.

3.2.3. Extraction of the objects that compose the price

Having the image with the components that are possibility to be the price, this stage aims to extract, among these components, those more similar to the price. This extraction is realized based on new filters, that eliminates the less similar components up to arrive in a components set that have the price feature.

The first filter is related with the components perimeter (this measure can be obtained by the “arcLength” function of OpenCV Library). Like the method used on the previous stage with the area, we observed that the numbers perimeter stands between 300 and 700. Therefore, we applied a thresold so that just the components with its perimeters in this range be selected, and the others discarded.

The second filter applied is related with the components dimension, more precisely with the height and width. Similarly, we noted that all numbers have height greater than 80 and width greater than 15. Therefore, we apply a threshold deleting the components that don’t apply the rule. After that, we compute the variance of

each component, based on the measures, because once the numbers have a similar shape, the variance is very good to show us what are the more similar components (Those that the variance is minimal). Therefore, we apply the Threshold based on variance, keeping just the objects that belong the price and the objects that are similar, in shape, to the numbers.

Lastly, we apply other filter to extract indeed the price. This filter is related with the distance between the elements. First we compute the distance between each component. From a distance vector, we verify the average of the distances and we delete the components that flee the average, until the average of the selected componets be lesser than 10000 (value computed observing the input images). Therefore, we reach the filter end, that basic on 95% of the images tested extract correctly just the price, like the next images.

3.2.4. Tesseract OCR Interaction

After all the processes described above have been realized, we can use the Tesseract OCR in order to obtain the String containing the price. But this project part have high difficult due to lack of understanding about the Tesseract. Like shown above, the numbers are perfectly visible and without noises, fact that would imply in the success on the OCR interaction, what did not always happen.

We did several tests, and we note that the Tesseract just recognize numbers when these are evident and with ideals shapes. What we cannot understand is the reason of the OCR don’t recognize the numbers in images seemingly perfect, don’t returning nothing in the String.

We tried to apply many approaches in order to do that the Tesseract recognize the price, such as: To take number by number and send to OCR, To apply morphological operations, To apply rotations, Filters, among other approaches, without the expected success.

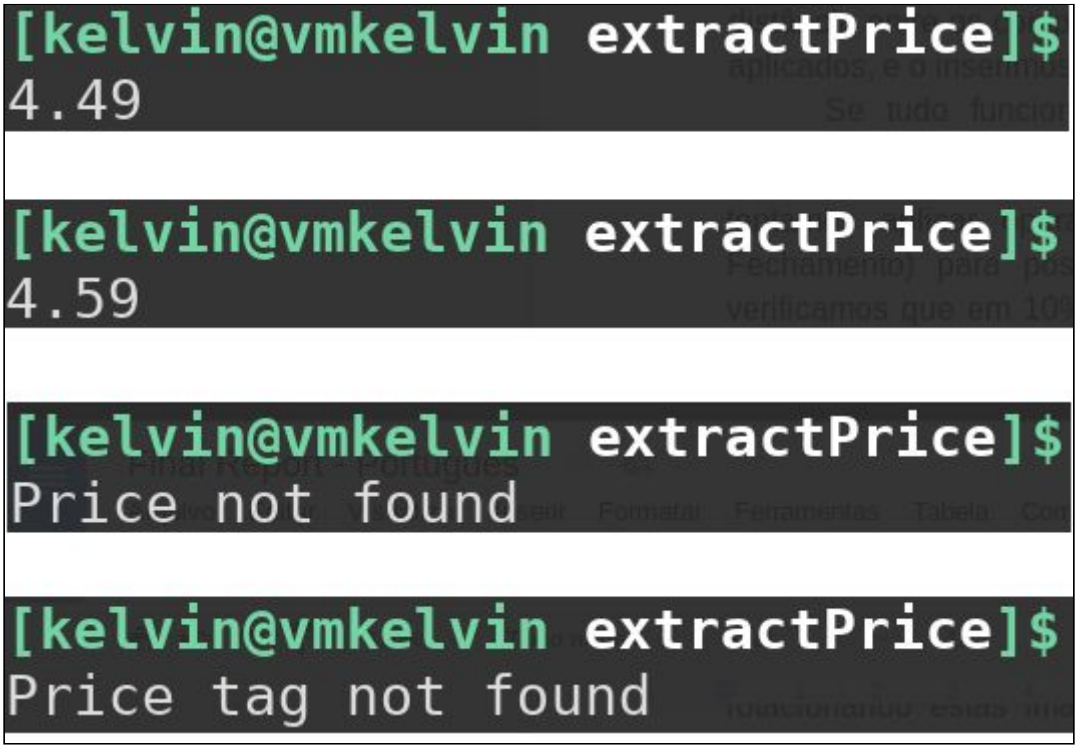
The main similarity between the most images that the Tesseract did not recognize is that all appear to be rotated. So we did a test, rotating all images so that stay aligned, without success, and after we took the images recognized, rotated them and, to our surprise, they continued working, like the image below.

So, we suppose that the failure with the OCR is because the perspective notion of the image, that the numbers become with a 3D notion, some more distant of the camara that others. But we could not prove this theory neither to fix that.

Finally, with the images that Tesseract recognizes, we compute the exact place of the price separation point (integer part, decimal part) by the distance between the price components, once that the point disappears with the applied filters, and we put it in the correct place on the String.

If all worked until now, we have the desired String. Otherwise, we try to apply morphological operators in the image (Erosion, Dilation and Closing) to after send to Tesseract. Based on tests, we verify that in 10% of images that the Tesseract would not recognize, after this approach, he starts to recognize.

Therefore, we finalize this step, finding the results like the next examples.



4. FINAL RESULTS

As seen previously, our software can be divided into two parts: one for location and extraction of the label and another for extraction of the price. The junction of these two parts was successful in just under half of the images taken (approximately 40%). This is due primarily to two reasons: price label location errors or errors in price price extraction (described earlier).

This accuracy value can be improved mainly in understanding how the Tesseract recognition software works, as often price tags were located, but the numeric values did not. In addition, mapping a better label localization strategy, how to use the idea of regions of interest can improve accuracy.

5. REFERENCES

Template matching
http://docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/template_matching/template_matching.html

Feature matching
http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_matcher/py_matcher.html

Morphological operations
http://docs.opencv.org/2.4/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html

Detection of objects by color
http://www.pyimagesearch.com/2014/08/04/opencv-python-color-detection/
http://www.pyimagesearch.com/2016/02/15/determining-object-color-with-opencv/

Tesseract
http://docs.opencv.org/trunk/d7/ddc/classcv_1_1text_1_1OCRTesseract.html
https://www.packtpub.com/mapt/book/application_development/9781785280948/10/ch10lvl1sec72/using-tesseract-ocr-library