Project Report
Product Recognition of Store Shelves

Mattia Campestri

Alberto Jesu

Gioele Pisanelli

Introduction

This report describes the practical implementation followed to solve the problem of object recognition on store shelves. Since the project was divided into different steps, related to tasks to perform on different subsets of images, each paragraph of this document will cover one of the steps/subsets.

1. Easy scenes

The first subset of scenes contains only a limited number of boxes, each present only one time, without repeated boxes and at a high enough resolution. For this scenario, the following pipeline was deployed.

1.1 SIFT feature detection and Flann matching

We decided to compute SIFT keypoints and descriptors for each box and each scene. The resulting features were then used to find matches between every box and scene using a K Nearest Neighbour Flann Matcher. Through the found matches we were then able to compute the homographies warping the box points into scene points, obtaining an initial guess of a potential found box. Considering the applicative scenario, to speed up the computation we decided to precompute the features for all the templates before starting the search in the scenes. In a real-world deployment, this operation could be executed automatically whenever a new product is inserted in the database of the supermarket, thus avoiding repeating the feature detection step at each new execution of the application. This optimization was used for all the subsets of scenes.

1.2 Match validation

The matching step produced some false positives, which were removed with the use of a series of checks. First, we made sure that the found polygon was convex, which of course represents a requirement for the shape of a box.

Then, we checked its rectangularity (the ratio between its area and the area of its bounding rectangle), deleting matches with a low rectangularity value, which would not represent a suitable shape for a box.

Finally, we addressed the problem caused by the presence of different variants of the same brand of cereals. Since the matching step is performed on grayscale images, colour differences between these boxes are lost, resulting in the occurrence of false positives. This was solved by comparing the colours of the template with the instance found in the scene, to test the compatibility of the match. This was done by computing the histogram of the hue channel of the two images, collecting the values into 8 bins, in order to group together similar colors. Then, the correlation of the two histograms was computed, as well as their peaks, which indicate the main colors of the images. Matches with a low correlation or without enough peaks in common are discarded. In order to make this last check as effective as possible, we tried to compare only the colours of the regions not containing many features. In fact, because of how the feature detection algorithm works, features belong to areas with edges and intensity variations, which in the case of grocery products are typically represented by drawings, logos, mascots and product names, which can be equal in different variants of the same product, thus producing wrong matches. Removing these regions from the color comparison step gives more reliable results, reducing the number of false positives. So we drew a black rectangle around each matching point found in the two images, obtaining a box and a scene in which only plain regions of the image that contain no important features are left, allowing us to use those only for colour validation. Moreover, an approximation of the percentage of the area of the image covered by features is computed. If this percentage is lower than a certain threshold, it means that only a small area of the image was matched, which typically happens when logos are matched between two different products of the same brand. In this case the result can be discarded to be sure to avoid a false positive.

Even if it consists of multiple steps, the evaluation process is very efficient and does not have a significant impact on execution time.

1.3 Results

Through this method, we were able to correctly identify all the boxes in each scene. For each found instance we record the bounding rectangle and the position of the barycenter. Figure 1 shows an example of the achieved results.

2. Medium scenes

The next subset of scenes shows a higher number of boxes, with the possible presence of multiple instances of the same box.

2.1 SIFT feature detection and Flann matching

In this case, we followed an initial approach similar to the one used for the previous subset, using SIFT and Flann KNN to obtain matches.

Figure 1. Example of box identification result in easy scene.

2.2 Generalized Hough Transform

We then decided to apply the Generalized Hough Transform using the SIFT keypoints and the barycenter of each box. We implemented the GHT algorithm in the following way:

1. Given a box to search in a scene, where we already obtained the matching keypoints and the barycenter of the box, we compute the joining vector from each keypoint to the barycenter.
2. Exploiting the fact that each SIFT keypoint has its scale and orientation, we resize and rotate each of the joining vectors by, respectively, the ratio of the scales and the difference of the orientations of the matching keypoints.
3. Then, by applying these vectors in the scene, we perform a voting process that gives more votes to the points that are more likely to coincide with the actual barycenter of the box in the image.
4. We then find the supposedly correct barycenter by performing a dilation to group the potential barycenters together, obtaining a blob whose center is the potential barycenter. We preferred using this method over any kind of clustering process since the latter would have been more complex and computationally expensive.
5. Finally, we compute the homography warping the box into the scene. For this we need at least 4 matching points. To find these points, we search for keypoints whose joining vector falls in the neighbourhood of the found barycenter, as these keypoints are more likely to be good matches.

We applied this procedure for all the boxes, obtaining many potential matches.

2.3 Match validation

We proceeded by applying the checks we introduced before (convexity, rectangularity, colour validation), to better filter the potential boxes. In this subset of scenes, however, this wasn't enough, since sometimes we found overlapping boxes: to solve this issue we implemented another check to avoid a box to be inside another. To make this efficient enough, we wanted to only compare two potential matches that seemed to overlap, instead of cycling through each possible pair of matches checking for intersection, and to carry out the checks while computing the potential boxes, in order to keep consistency during the whole search process. To do so, each time we found a new potential box, we drew its barycenter on a mask with the same shape of the scene, using a different gray level for each new barycenter. By doing so, we record the barycenter of each new match on the mask, and can look for intersections between the currently drawn barycenter and the previously drawn ones using a simple bitwise and operation between masks and extracting the gray level of the previous intersected barycenter. This gray level is then used as a key to access the bounds associated with the intersecting barycenter, which were previously stored in a dictionary. Having obtained a potentially overlapping pair, we then applied the check and saved in the dictionary at the particular intensity only the correct box, or both boxes if they weren't actually contained inside each other.

```
new_barycenter_mask = np.zeros(matches_mask.shape, dtype=np.uint8)
cv2.circle(new_barycenter_mask, (cx, cy), w // 4, color, -1)
new_bar_copy = new_barycenter_mask.copy()
new_bar_copy[new_bar_copy > 0] = 255
intersection = cv2.bitwise_and(new_bar_copy, matches_mask)

if cv2.countNonZero(intersection) > 0:
    gray_index = intersection[intersection > 0][0]
    bar_intersected = bounds_dict[gray_index]
    bar_intersecting = (new_bounds, test_box, box_name)
    result = compare_detections(bar_intersected, bar_intersecting)
   ...
```

Figure 2. Excerpt of the implementation of the intersection check.

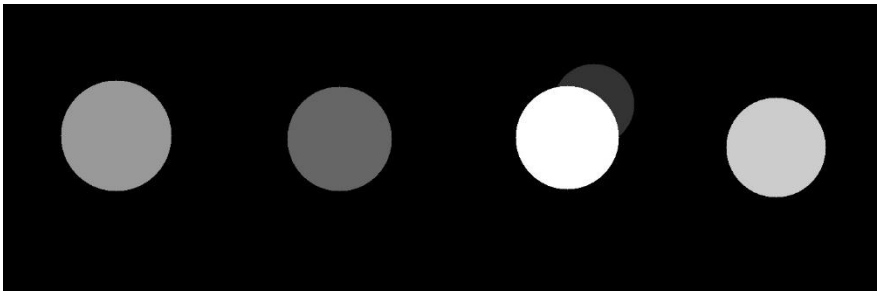

Figure 3. Mask used to find intersections between matches. The two intersecting circles show a conflict between two candidates.

Figure 4. Example of box identification in medium scene.

2.4 Results

At the end, the resulting matches are found in the dictionary mentioned in the previous section. For each found instance, we report the same information as in the easy scenes subset and draw the matches on the scene. The adopted strategy yields good results, correctly finding all the cereal boxes in each scene. An example of the obtained results is represented in Figure 4.

3. Hard scenes

The images in the last subset represent multiple shelves, each containing many cereal boxes, resulting in total number of around 40 boxes. An additional obstacle to the detection process is represented by the low resolution of the images, which causes each box to take up a relatively small number of pixels. In this scenario we decided to use the same approach as with the medium scenes, but with some modifications to obtain a higher number of matches and to improve the execution time.

3.1 Shelves splitting

Given the setting, we decided to leverage some domain information to facilitate the detection task and reduce the processing time. This is done by splitting the scene image using the horizontal separators between shelves. This way we obtain a sub-image for each shelf and we can execute the same steps as seen before on a smaller portion of the image, reducing the complexity of the

Figure 5. Lines found by the shelves splitting algorithm.

detection process and enabling parallel processing of each sub-image. The splitting step is executed by thresholding the whole scene using fixed HSV thresholds that were empirically found to match the values of the separators between the shelves. Afterwards, we used the Hough Transform for lines detection to find horizontal lines, which were then refined to find a single line for each division between two shelves. An example of the resulting division is shown in Figure 5.

We then proceeded by creating a pool of processes and assigning to each process one of the obtained sub-images, on which the same pipeline as with the medium scenes is executed (SIFT feature detection, Flann feature matching, Generalized Hough Transform, match validation). This parallelization results in a significant speed-up, as shown in figure 6.

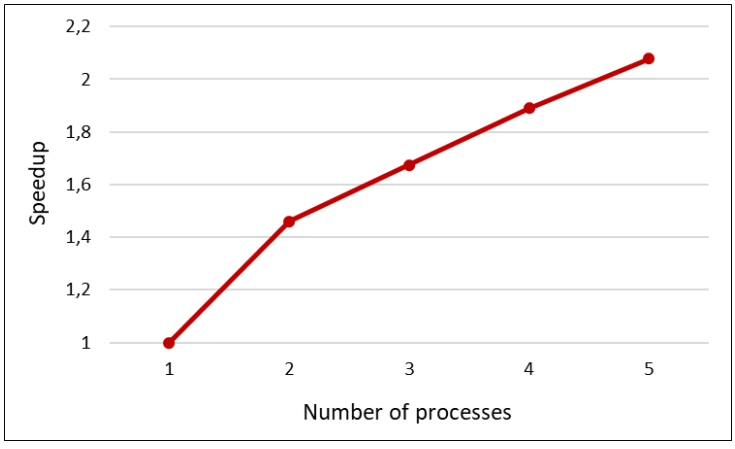

Figure 6. Speedup achieved by processing shelves in parallel.

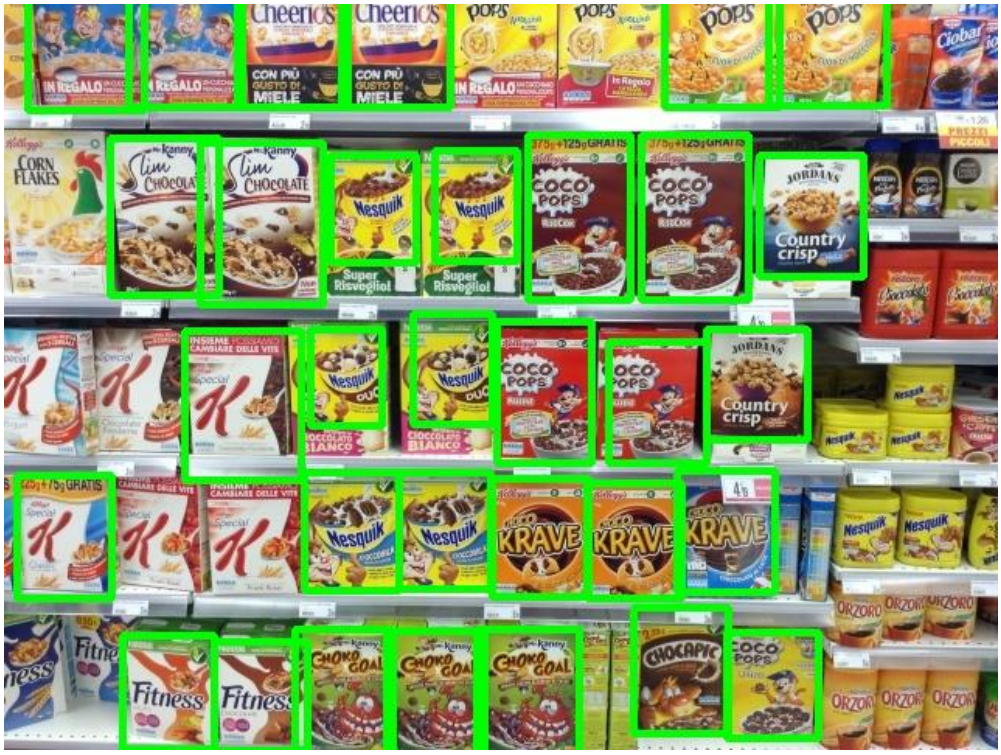

Figure 7. Example of box identification in hard scene.

3.2 Increasing matches number

Given the low resolution of the images, we obtained better results by upscaling the sub-images to double their size and by blurring the larger templates before feature detection. Moreover, in order to obtain a higher number of matches, we used a higher ratio threshold in the Lowe ratio test after Flann feature matching. For the same purpose, we also relaxed some of the constraints of the match validation step, using more permissive thresholds in order to allow more matches at the expense of a small amount of precision.

3.3 Results

In the end we obtained results such as the one in Figure 7, in which we found a good number of boxes, although we encountered some problems:

- We couldn't recognize all boxes present in the various scenes. This is probably due to the low resolution of the scenes, as well as some of the template boxes not being identical to the ones depicted in them, lowering the number of matches we could find.
- In some cases, namely the red and brown *Coco Pops* boxes, the hue validation process failed to distinguish one from the other, as the brown and red colours share most of their hue values.

4. Conclusions

As discussed in the previous section, some improvements could still be made, especially in the case of scenes containing multiple shelves. In this case, in particular, we had to find a balance

between false positives and missed matches, in order to recognize a sufficient number of boxes without sacrificing precision too much. The shelves splitting phase could be improved as well, in order to make it more robust against lighting changes that could create reflections in the separators, thereby hindering the ability to make correct splits. However, despite the highlighted flaws, the obtained results are overall satisfying, as the developed solution correctly recognizes most of the boxes in the majority of the scenes.