PRODUCT RECOGNITION ON STORE SHELVES

Image Processing and Computer Vision

Full Name	**Student ID**
Ancarani Elisa	1026272
Deana Alessia	1039580

Academic Year 2021 - 2022

Contents

1 | Introduction

1.1 | Abstract

In this project, computer vision techniques are implemented in order to perform object detection. Specifically, the aim is to implement a program for product recognition on store shelves. The aim is to recognise boxes of cereals of different brands located on the store shelves, and, respectively, their position and the number of instances of the same boxes appearing in the scenario. In this report, it is explained the solution proposed using OpenCV.

1.2 | Task description

This project consists of two main tasks and an optional task. To perform all the tasks we are given a set of model images, one for each product and a set of scene images.

First task (step A) is the simplest and consists of identifying a single instance of a product in a scene image. In addition to what achieved in the previous task, in the second (step B), it is required to detect multiple instances of the same product in a given scene image. To obtain this goal, it is proposed to use local invariant features together with Generalised Hough Transform.
Finally, in the last task (step C), given the scenarios with more than 40 products per image, it is required to identify as many instances as possible.

2 | STEP A

Object detection pipeline implemented for solving task A consists of the following steps:

- Identification of key points in model and scene images.
- Matching key points using FLANN algorithm.
- If enough matches are found, they are used to estimate an homography using RANSAC.
- Color and dimension filtering.

In first point, it is initialised a SIFT (scale-invariant feature transform) object:
`> sift = cv2.xfeatures2d.SIFT_create()`

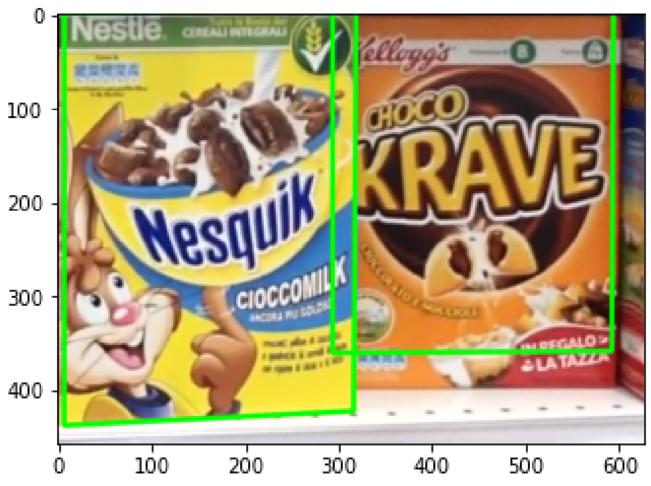
A SIFT feature is a key point (i.e salient repeatable point) with an associated descriptor. A SIFT key point is described by four parameter: x and y coordinates, its scale (radius of the region) and orientation (angle in radians). SIFT descriptor allows to detect characteristic features, structures in the image. Searching for them at multiple positions and scales, SIFT descriptor is rotation, translation and scale invariant.

Once we have computed the SIFT descriptors for each reference image we find the matching between keypoints using an approximate kd-tree algorithm from **FLANN** (*Fast Library for Approximate Nearest Neighbors*) included in OpenCV. In Python can be implemented by getting a `FlannBasedMatcher` object and calling the method `knnMatch` with the two sets of descriptors of the model and the scenario passed as parameters. The two best matches for each keypoint are found using Lowe's ratio test function (`lowe_ratio_test`), which checks that the two distances are sufficiently different: if they are not, then the key point is eliminated and will not be used for further calculations. Otherwise, you go through the `compute_mask()` function, where the masks for each model in the scene are calculated as follows:

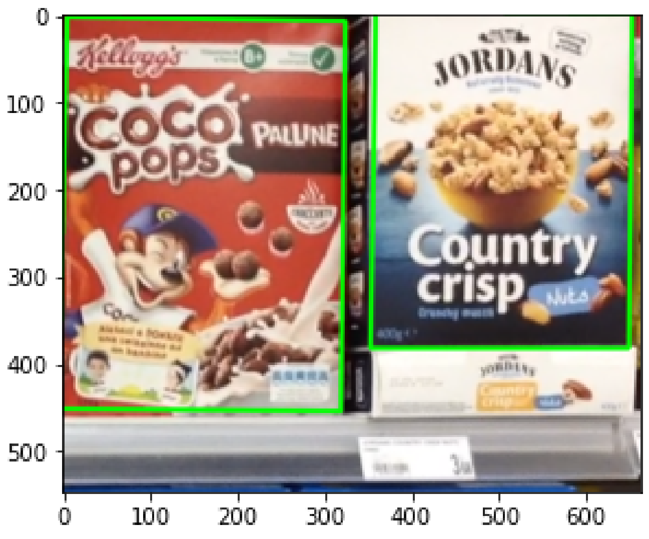
A minimum number of matches (`MIN_MATCH_COUNT`) have to be there in order to find the model under analysis in the scene image. If enough matches are found, locations of matched keypints are extracted in both the images building a correspondences arrays of good matches, these correspondences arrays are `src_pts` and `dst_pts`. Through OpenCV it is possible to estimate a robust homography by mapping the key points of the model to the corresponding key points in the scene, using RANSAC. The input parameter of RANSAC are the correspondences arrays of good matches.
Finally, for colour filtering we have calculated the distance between the colour mean of the model and the colour mean of the different model in the scene, if this distance is less than a certain colour threshold, we proceed doing proceed with box dimension filtering, checking the goodness corners, in order to improve the quality of the masks drawn by continuing to discard false positive instances.

2.1 | Output example of task A:

```
Product 0 - 1 instance/s found
     Position (163, 215), width:  310 px, height:  433 px
Product 11 - 1 instance/s found
     Position (444, 167), width:  299 px, height:  388 px
Product 19 - 0 instance/s found
Product 24 - 0 instance/s found
Product 25 - 0 instance/s found
Product 26 - 0 instance/s found
```



```
Product 0 - 0 instance/s found
Product 1 - 0 instance/s found
Product 11 - 0 instance/s found
Product 19 - 1 instance/s found
     Position (504, 190), width:  295 px, height:  384 px
Product 24 - 0 instance/s found
Product 25 - 1 instance/s found
     Position (161, 229), width:  321 px, height:  447 px
Product 26 - 0 instance/s found
```



3 | STEP B

Step B can be summarised in the following steps:

- Identification of key points in model and scene images.
- Matching key points using FLANN algorithm.
- Implementation of the Generalised Hough Transform (GHT).
- If enough matches are found, they are used to estimate an homography using RANSAC.
- Color and dimension filtering.

Step B is implemented starting from step A, what is meaningful is the creation of the GHT. To detect edges we have always relied on SIFT features, taking as reference point the barycentre of the good key point that passed the Lowe's ratio test. Initially, function `get_hough_space()` allows to find the elements of the Hough Space: `x_center, y_center, scale_change, translation`.
Translation between two features is calculated by taking subtraction of the barycentre of model key point with the coordinates of the key point. The implemented GHT is scale invariant. Instead, rotation is not taken into account because of the presence of only upright cereal boxes.

For every model in the scene, the barycentre is calculated and the function `hough_elements()` is called, in which we passed as parameter the barycentre (`mean_xc, mean_yc`), and the key point of the scene and the model. This function is build by cycling along the good matches, stored in `good_dict`, that passed the Lowe's ratio test and it is build a Hough space dictionary in which we compute the dimensions of the Hough space by calling the `get_hough_space()` function.

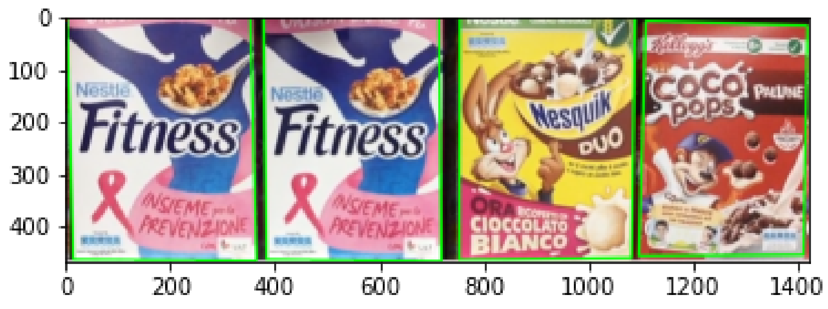
In order to calculate each bin of the Hough space the `calculate_bin()` function has been implemented, which compute the bin size given the scaling change of the Hough space values. The function returns x and y bins and their related *size*.

After the above matching process, we come up with a set of matches then the voting process is performed: we vote into the accumulator array by increasing the count of the cells. Thus, peaks using local maxima are found in the accumulator array and only those in which the count is greater than 5 are considered.

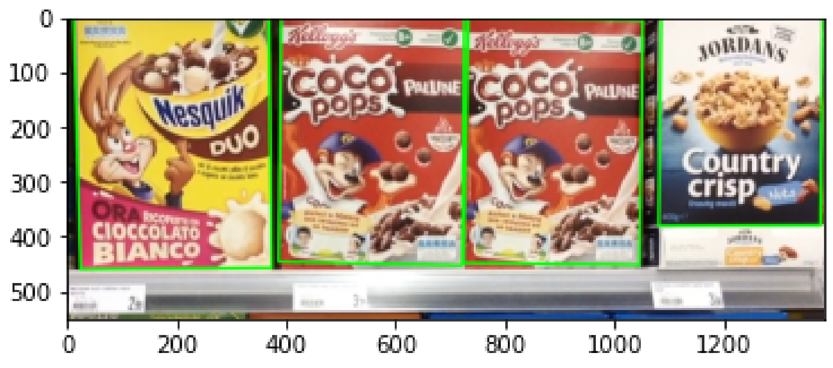
Finally, we build a dictionary, `s_global_correspondences`, in which we insert for each model the array accumulator and the key points related to the dictionary of *good matches* returned with Lowe's ratio test. Through the function `compute_mask`, we proceed to the colour and dimension filtering as in step A, not taking into consideration the dictionary of good matches but the one calculated previously (`s_global_correspondences`).

3.1 | Output example of task B:

```
Product 24 - 1 instance/s found
     Position (184, 214), width:  351 px, height:  500 px
Product 24 - 1 instance/s found
     Position (550, 219), width:  335 px, height:  489 px
Product 26 - 1 instance/s found
     Position (921, 218), width:  333 px, height:  486 px
Product 25 - 1 instance/s found
     Position (1261, 233), width:  312 px, height:  443 px
```



```
Product 26 - 1 instance/s found
     Position (195, 206), width:  353 px, height:  505 px
Product 25 - 1 instance/s found
     Position (558, 222), width:  335 px, height:  456 px
Product 25 - 1 instance/s found
     Position (890, 228), width:  320 px, height:  445 px
Product 19 - 1 instance/s found
     Position (1233, 190), width:  295 px, height:  383 px
```



4 | STEP C

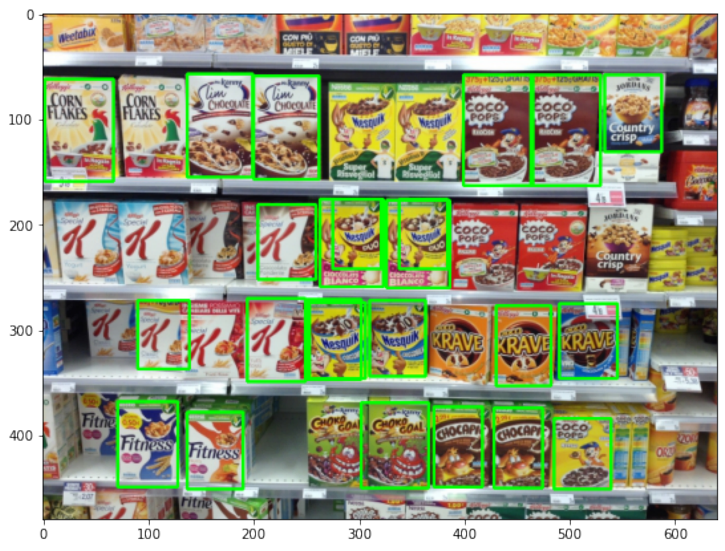
The goal in step C is to try to identify as many products as possible in the different scenarios. We have decided to change technique, different from the previous ones, which allowed us to further learn new methods of approaching this issue. The problem was to extract smaller cereal boxes from large shelves, so we thought about the **template matching technique**, whose goal is to find a smaller image into a larger image. To perform template matching, we have always used the OpenCV library.
Strategy implemented for solving the task can be summarised in the following steps:

- Resize of models: if the height of the model is greater than 1000 pixels then the size of the model image is reduced to 50%.
- Call the templateMatch function and get the rectangles. Here's the operation:
  - For each model we take 40 different scale levels from `0.05` to `0.5`.
  - At each scale level the model image is resized by calling the **`resize`** function which maintains the aspect ratio provided by **`imutils`** library.
  - Calculate **`matchTemplate()`** function using the `cv2.TM_CCOEFF_NORMED` method which gives maximum value as the best match.
  - Use the **`minMaxLoc()`** function to found the best matches as global minimums or maximums.
  - The **`best_match`** variable is filled with the first value of the global maximum **`(max_val_t)`** and then the first **`max_val_t`** is compared with those calculated for each scale level to consider only the best one. So we also need to acquire the height and width from test resized model.
  - Use two conditions to get rid of incorrect detections (i.e. **`length < 45`** and **`max_val t >=`** `0.4)`. Then resize the model, consider the best scale level, and compute the **`matchTemplate()`** function. Considering the global maximums we take only the points of rectangles that represent the masks greater than a fixed threshold.

Template matching technique is by definition translation invariant, but not scale and rotation invariant. Above has been proposed an extension by resizing the model according to the scale. Template matching can make few mistakes in detecting some instances this because they present general features that are similar in certain models. Template matching is not ideal when dealing with rotated object, in this case is better use key point matching. However, for this specific application, we obtain fairly satisfactory results because we have to deal only with upright products.

4.1 | Output example of task C:

```
Product 0 - 2 instance/s found
     Position (310,272), width:  53 px, height:  72 px
     Position (253,274), width:  53 px, height:  72 px
Product 1 - 1 instance/s found
     Position (490,275), width:  55 px, height:  71 px
Product 2 - 1 instance/s found
     Position (2,62), width:  65 px, height:  97 px
Product 3 - 1 instance/s found
     Position (302,369), width:  64 px, height:  81 px
Product 4 - 2 instance/s found
     Position (137,57), width:  62 px, height:  98 px
     Position (200,59), width:  62 px, height:  98 px
Product 5 - 1 instance/s found
     Position (251,271), width:  51 px, height:  76 px
Product 6 - 2 instance/s found
     Position (275,176), width:  46 px, height:  66 px
     Position (338,176), width:  46 px, height:  66 px
Product 7 - 2 instance/s found
     Position (399,58), width:  66 px, height:  105 px
     Position (463,58), width:  66 px, height:  105 px
Product 8 - no instance found
Product 9 - 1 instance/s found
     Position (90,272), width:  49 px, height:  65 px
Product 10 - 1 instance/s found
     Position (204,181), width:  55 px, height:  71 px
Product 11 - 1 instance/s found
     Position (430,277), width:  52 px, height:  76 px
Product 12 - 1 instance/s found
     Position (137,378), width:  53 px, height:  73 px
Product 13 - no instance found
Product 14 - 2 instance/s found
     Position (371,373), width:  46 px, height:  76 px
     Position (428,374), width:  46 px, height:  76 px
Product 15 - 1 instance/s found
     Position (485,384), width:  54 px, height:  67 px
Product 16 - no instance found
Product 17 - no instance found
Product 18 - no instance found
Product 19 - 1 instance/s found
     Position (533,58), width:  54 px, height:  73 px
Product 20 - 1 instance/s found
     Position (194,270), width:  55 px, height:  79 px
Product 21 - no instance found
Product 22 - no instance found
Product 23 - 1 instance/s found
     Position (90,272), width:  49 px, height:  65 px
Product 24 - 1 instance/s found
     Position (71,368), width:  57 px, height:  81 px
Product 25 - no instance found
Product 26 - 2 instance/s found
     Position (263,176), width:  60 px, height:  83 px
     Position (326,177), width:  60 px, height:  83 px
```


0
100
200
300
400
0
100
200
300
400
500
600

A | References

Professor Luigi Di Stefano slides

Professor Luigi DI Stefano laboratories 2021-2022

https://docs.opencv.org/4.x/d4/dc6/tutorial_py_template_matching.html

https://pyimagesearch.com/2015/01/26/multi-scale-template-matching-using-python-opencv/

https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html