Product recognition

Computer Vision Project Report

Michele Calvanese
Vincenzo Collura
Samuele Marino

University of Bologna
Italy
May 2022

Contents

1 Introduction

The project we have chosen is *Product Recognition on Store Shelves* whose goal is to create, given the image of a store shelf, such a system should be able to identify the different products (in this case the material of the project contains only boxes of cereals) present, their size and their position. The project is composed of three different tasks which will explain the development and the strategies used in their resolution. Our code together with the model and scene images can be found here.

1.1 Overview of the setup

The project was implemented thanks to the `OpenCV` library [1]. The core of the setup is in the `utils` folder. To find the potential bounding boxes between the models and the scene, two classes have been created:

- `FeatureMatcher`: uses local invariant features with `SIFT`, and computes the homography between the model and the scene by applying the *RANSAC* method on the matching keypoints. It only detects single instances of the model, and is explained in section 2;
- `MultipleInstanceMatcher`: it extends `FeatureMatcher` by applying the *Generalized Hough Transform* on the matching keypoints using the star model. It uses the scale and rotation information found with `SIFT`. It is designed to find multiple instances of the model. It is explained in section 3.

These two classes can be found in the matchers.py file. Additional insights can be found in the notebook `workflow.ipynb`. The `bbox_filtering.py` file contains all the functions used to filter the potential bounding boxes given by the homographies (using color, shape and overlap between bounding boxes). This is explained in more detail in section 4. The file `visualization.py` contains all the functions to visualize the result. Everything was done and explained in notebook `product_recognition.ipynb`.

2 Single instance detection (Step A)

This part is performed by `FeatureMatcher`. Its structure can be seen in figure 1.

2.1 Brief description

The goal is to identify single instances of reference images in a scene image.

To solve this task we use local invariant features, as seen in the sixth laboratory and as suggested by the project assignment itself. For this we use the `SIFT` algorithm. After finding the matches of the keypoints in the scene and the model images, a homography between the two images is computed. The potential bounding box is then inferred from the homography.

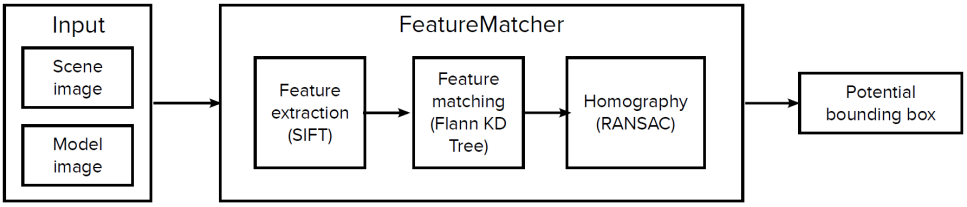

Figure 1: Scheme of the matching algorithm in `FeatureMatcher` for single instance detection.

2.2 Computation

2.2.1 Feature extraction (SIFT)

We first initialize the `SIFT` Detector of `OpenCV`, which allows us to calculate the keypoints and descriptors of scenes and models. In order to avoid to compute the same keypoints multiple times when detecting multiple models in a scene (or in multiple scenes), the keypoints and descriptors can be passed from the outside. Once the descriptors for each keypoint in the model and scene image have been calculated, we need to find the matching keypoints.

2.2.2 Feature matching (FLANN KD-Tree)

To do this we use the FLANN KD-tree, which is an indexing technique borrowed from database management, to speed up the search for matches. This algorithm is taken from `FLANN` included in `OpenCV` and contains a collection of algorithms optimized for fast nearest neighbor search. Now we are filtering false matches in `SIFT`. To filter the matches, we use a distance ratio test to try to eliminate false matches. The distance ratio between the two nearest matches of a considered keypoint is computed and it is a good match when this value is below a threshold set by us. This ratio helps to discriminate between ambiguous matches and well discriminated matches.

The distance ratio can be passed to `FeatureMatcher` as `match_distance_threshold`. Its default value is 0.7.

2.2.3 Homography (RANSAC)

To find the position we compute, given the correspondences, the homography between the matching keypoints in the model image and in the scene image. However, since spurious matches are still present, computing it using the least squares method leads to imprecise results. So we compute the homography using *RANSAC* (RANdom SAmple Consensus) [2], an algorithm designed to fit a parametric model to noisy data. In our case we estimate the homography from the good matches, while identifying and discarding the wrong ones. The default threshold for the RANSAC reprojection error is set to 1 px.

2.3 Potential bounding box

From the homography, a potential bounding box is computed by applying the homography on the corners of the model image. This potential bounding box must then be validated by the filters explained in section 4.

3 Multiple instance detection (Steps B and C)

This part is performed by `MultipleInstanceMatcher`, which extends `FeatureMatcher`. Its structure can be seen in figure 2.

3.1 Brief description

In this second step, the feature extraction and feature matching parts are identical to those explained in section 2. In addition to what was done there, we used the *GHT* (Generalized Hough Transform) to detect multiple instances of the same model. After the GHT, a homography is computed for every instance of the model image.

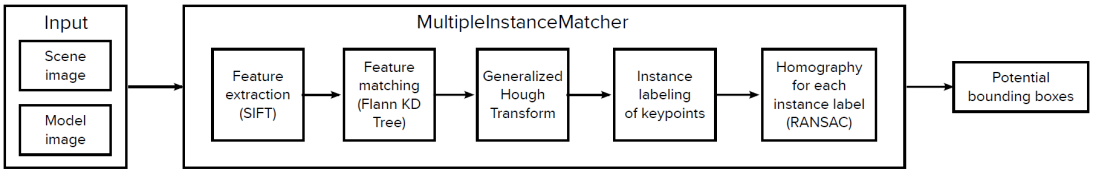

Figure 2: Scheme of the matching algorithm in `MultipleInstanceMatcher` for multiple instance detection.

3.2 Computation

3.2.1 Generalized Hough Transform

After using `SIFT`, the R-vectors are computed. These are defined as the difference between the center of gravity of the model (which is just the average position of the keypoints) and the keypoints. The rotation and scale factor are inferred from the `SIFT` descriptors.

Now it's time to vote for the barycenter position in the scene. To do this, the R-vectors are rotated and scaled first, and then each vote is calculated as the vector sum between the positions of the matching keypoints in the scene and the corresponding R-vectors. The votes are then collected in an accumulator array. The accumulator array casts the votes in bins of size K, which is a parameter that can be set externally. We chose $K = 15$ as the default value.

3.2.2 Instance labeling of keypoints

After computing the accumulator array, its local maxima are found using `find_peaks` from `scipy.signal` [3]. To be considered a local maximum of the accumulator array, we chose that a peak needs to have at least 30 % of the height of the global maximum.

To be able to divide the various instances of the same model, we assign different labels to each keypoint. To do this, we calculate the distance between each peak in the accumulator array and the vote corresponding to each keypoint. The keypoints are thus labeled with the nearest peak to their vote position.

3.2.3 Homographies (RANSAC)

A homography is then computed for each label group, and the corresponding bounding boxes are inferred. This process works the same as in section 2.2.3.

3.3 Potential bounding boxes

As in section 2.3, the resulting potential bounding boxes must be validated by the various filters explained in section 4.

4 Bounding box filtering

The bounding box filtering is performed by `find_bboxes` in the `bbox_filtering` module in `utils`. The thresholds for each filter are passed as parameters to `bbox_filtering`.

4.1 Brief description

This part is the same for both single and multiple instance detection. It is performed after all potential bounding boxes of all the models in a scene have been computed. These potential bounding boxes are filtered with several filters. The bounding boxes that pass the filtering step are condidered as valid. Each filter is explained in detail in the next section. The pipeline of

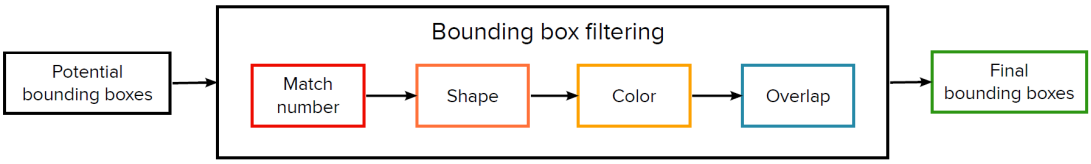

Figure 3: Filtering of the potential bounding boxes. The color of the filters corresponds to the color used for the visualization.

the filtering process can be seen in figure 3, and an example of the filters in action can be seen in figure 4.

4.2 Descriptions for each filter

4.2.1 Match Number

Bounding boxes that are obtained with a homography computed on a number of matches below a certain threshold are discarded.

The minimum threshold is given by the `min_match_threshold` parameter. Its default value is 15. The minimum possible value is 4, because at least 4 points are needed to find a homography.

4.2.2 Shape

This filter discards the bounding boxes that have an irregular shape. It first checks if the bounding box polygon does not have crossing between its edges. It then checks the ratio between both pairs of opposite edges and of the diagonals of the bounding box, after having sorted each pair in decreasing order. If any one of these ratios is higher than a certain threshold, the bounding box is discarded.

This threshold is given by `max_distortion`. Its default value is 1.4. The orange bounding boxes in figure 4 have been filtered by shape.

4.2.3 Color

The average RGB colors of the model image and the scene image in the bounding box are computed. These are then converted to the HSV color space, and their distance is computed. The distance of two colors c_1 and c_2 in the HSV color space is given by:

$$\begin{aligned} d_{\text{HSV}}\left(c_1, c_2\right) = &\Big(\sin(H_1)\, S_1\, V_1 - \sin(H_2)\, S_2\, V_2\Big)^2 + \\ &\Big(\cos(H_1)\, S_1\, V_1 - \cos(H_2)\, S_2\, V_2\Big)^2 + \\ &\Big(V_1 - V_2\Big)^2, \end{aligned} \tag{1}$$

where H_i, S_i and V_i represent respectively hue, saturation and value. If the distance between the two average colors is higher than a certain threshold, the bounding box is discarded.

This threshold is given by `color_distance_threshold`. Because the maximum value of d_{HSV} is 2, for ease of use it is first multiplied by a factor of 100 before being compared to the threshold. The default threshold value is 5. The yellow bounding boxes in figures 4 and 6a have been filtered by color.

4.2.4 Overlap

This filter checks every possible pair of bounding boxes to see if there are any overlaps. Given the sets of points of two bounding boxes A and B, the overlap coefficient c is given by:

$$c = \frac{A \cap B}{\min(A, B)}. \tag{2}$$

If for a pair of bounding boxes c surpasses a certain threshold, the bounding box with the least amount of matching keypoints is discarded.

The maximum threshold for c is given by `max_overlap`. Its default value is 0.8. The blue bounding boxes in figure 4 have been filtered by overlap.

Figure 4: Examples of filtererd bounding boxes on image `h1.jpg`. This is a filter demonstration run, so only the models `0.jpg`, `4.jpg` and `11.jpg` were used. The orange bounding boxes were filtered by shape, the yellow ones by color and the blue ones by overlap. The threshold values are the default ones. The green bounding boxes have passed all the filtering steps.

5 Image preprocessing and augmentation

For Task C we have to try to detect as many products as possible in this challenging scenario: more than 40 different product instances for each scene image, distracting elements (e.g. price tags, etc.) and low resolution images. The system described thus far has difficulty in finding enough matches between the models and these low-resolution images. This makes it necessary to use some kind of preprocessing.

For the model images we apply Gaussian filters of different widths: 1 px, 3 px and 5 px. Additionally, we resize the unfiltered model images to have a width of 360 px. For every model, all its different versions are labeled the same and used for the object detection. If multiple versions of the same model are found in the scene it is not a problem, because the overlap filtering in 4.2.4 takes care of that.

The scene images are upscaled by a factor of 4 using a neural network based approach, which is implemented in the `Super Resolution` module of `OpenCV`. We tried four different models on a laptop equipped with an Intel Core i7 8750H:

- FSRCNN [4]: fastest processing time (1 s) and worst performing;
- ESPCN [5]: almost as fast as FSRCNN (1.5 s), but significantly better than FSRCNN;
- EDSR [6]: slowest processing time (8 min) but best performance;
- LapSRN [7]: intermediate processing time (35 s) and performance.

The processing times refer to the preprocessing of all 5 scene images (see section 6.3).

6 Results

6.1 Task A

In Task A, we use the following model images:

`0.jpg, 1.jpg, 11.jpg, 19.jpg, 24.jpg, 25.jpg, 26.jpg`,

and the scene images `e1.png` to `e5.png`. In these scene images, only single model instances are present.

We use the default filtering threshold values and no image preprocessing. Except for `e3.png`, using `FeatureMatcher` for the computation of the homographies (as explained in section 2) leads to a perfect detection (see figure 5 as an example).

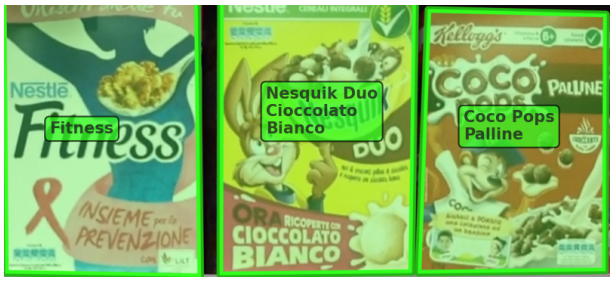

Figure 5: Results for task A on `e2.png` using `FeatureMatcher` for single instance detection.

The reason for the wrong detection on `e3.png` lies in the Choco Krave boxes (models `1.jpg` and `11.jpg`). `SIFT` is invariant with respect to color. This means, that for each Choco Krave model, the matches are about equally distributed between both Choco Krave boxes in the scene.

Therefore, when computing the homography, RANSAC will randomly choose one of the two boxes as the right one. If the wrong box has been chosen, the resulting bounding box is discarded when filtering by color. Depending on which bounding box was computed for which model, it can happen that either both models are correctly identified, one of the two is identified or none is. See figure 6a for an example in which none of the two models were correctly detected.

This problem is completely solved by using the `MultipleInstanceMatcher` class instead of `FeatureMatcher`. This is because effectively, for `SIFT`, this is a multiple instance detection problem. Using the GHT, the two Choco Krave boxes are separated: for each model image, two instances are found in the scene and two homographies and bounding boxes are computed. The wrong bounding boxes are subsequently filtered by color, and a consistent correct detection is thus possible (see figure 6b).

(a) **(b)**

Figure 6: Results for task A on `e3.png`. In (a) we used `FeatureMatcher`, and in (b) we used `MultipleInstanceMatcher`. The bounding boxes filtered by color are shown in yellow, the correct ones are shown in green.

6.2 Task B

In Task B, we use the same model images as in Task A. For the scene images we use `m1.png` to `m5.png`. We use `MultipleInstanceMatcher` and no image preprocessing. The filtering thresholds are the default ones. This setup achieves a perfect detection score. An example can be seen in figure 7.

Figure 7: Results for task B on `m4.png`.

6.3 Task C

In Task C, we use all the 27 model images. For the scene images we use `h1.jpg` to `h5.jpg`. We use `MultipleInstanceMatcher` and image preprocessing. To increase sensitivity, the filtering thresholds were modified as follows:

- `min_match_threshold`: 4;
- `color_distance_threshold`: 15;
- `bbox_overlap_threshold`: 0.5.

In addition, we increased `match_distance_threshold` (see 2.2.2) to 0.88 to have more matching keypoints.

As can be seen in figure 8, the image preprocessing greatly improves the detection sensitivity. The Special K boxes on the bottom left are not detected because of perspective distortion, which affects the performance of SIFT. In addition, the model images of these boxes are slightly different than the boxes in the scene, which reduces the number of matching keypoints.

Slight improvements in detection accuracy and sensitivity could be achieved with better parameter tuning or by trying different local invariant features algorithms.

(a) **(b)**

Figure 8: Results for task C on `h5.jpg`. In (a) no preprocessing on the models nor on the scene is performed. In (b), the scene image was upscaled by a factor of 4 using the EDSR model. The model images were filtered with Gaussian filters of 1.5 px, 3 px and 5 px, and an unfiltered version was resized to have a width of 360 px.

Bibliography

References

[1] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[2] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, jun 1981.

[3] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[4] Chao Dong, Chen Change Loy, and Xiaoou Tang. Accelerating the super-resolution convolutional neural network. *CoRR*, abs/1608.00367, 2016.

[5] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P. Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. *CoRR*, abs/1609.05158, 2016.

[6] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. *CoRR*, abs/1707.02921, 2017.

[7] Wei-Sheng Lai, Jia-Bin Huang, Narendra Ahuja, and Ming-Hsuan Yang. Fast and accurate image super-resolution with deep laplacian pyramid networks. *CoRR*, abs/1710.01992, 2017.