# PRODUCT RECOGNITION ON STORE SHELVES

**MANUEL GABRIELLI**

Computer Vision and Image Processing

—

University of Bologna

Two Years Master in Computer Engineering

## 1 - OVERALL TASK

The Overall Task is to develop a computer vision system that, given a reference image for each product, is able to identify boxes of cereals of different brands from one picture of a store shelf.

For each type of product displayed in the shelf the system should report the number of instances, the dimension of each instance and its position in the image reference system.

## 2 - STEP A – MULTIPLE PRODUCT DETECTION

The goal of Step A is to recognize a single instance of a product within an image. The set of products consists of five images, called models, each depicting a single box of cereals, while the set of scenes, depicting the shelves in which the products are stored, consists of seven images. The purpose of the system is to be able to identify each product placed on the shelves.

This task is solved by using local invariant feature Paradigm. The steps taken to obtain the results are visible in Figure 1. All the steps listed are performed for each given scene.

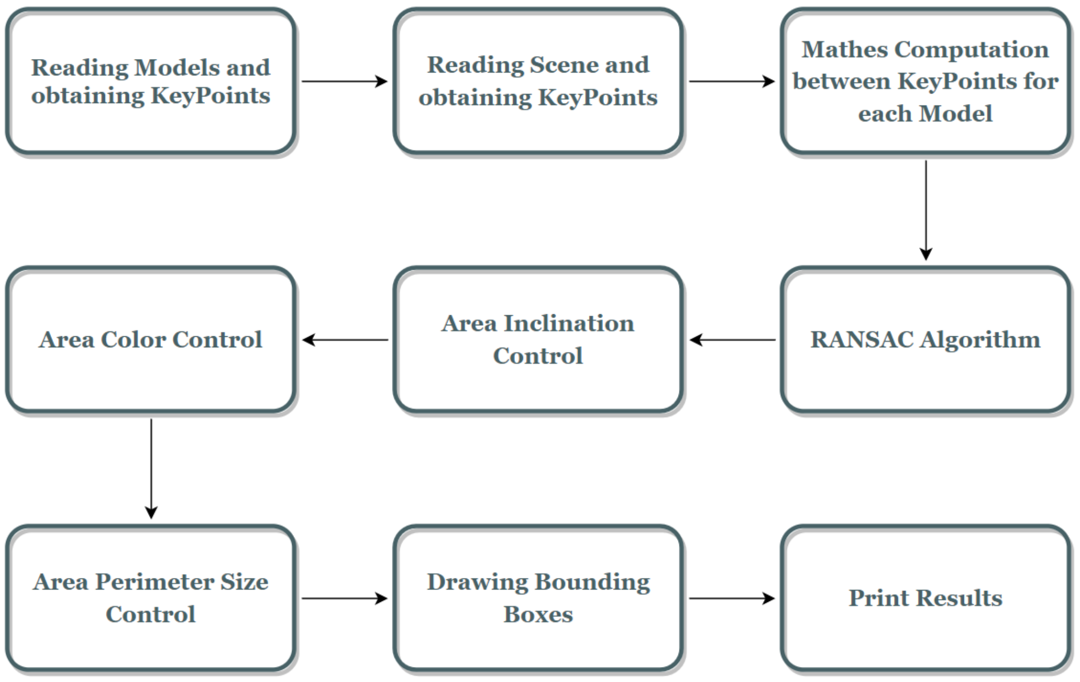

Reading Models and obtaining KeyPoints
Reading Scene and obtaining KeyPoints
Maths Computation between KeyPoints for each Model
RANSAC Algorithm
Area Inclination Control
Area Color Control
Area Perimeter Size Control
Drawing Bounding Boxes
Print Results

Figure 1 – Step A Diagram

## 2.1 - Implementation Step A

First, all the images of the models and scenes to be analyzed in this step are read. All these images are then converted from BGR to RGB and saved in special structures to subsequently exploit them.

Then the average color of each model is calculated, but not on the entire box but by eliminating the first third of the image which, in most cases, contains the logo of the cereal brand, similar in several models.

Next, an instance of the SIFT detector is instantiated to obtain the KeyPoints of each model. Once obtained through the *getKeyPoints* function, they are saved in a special dictionary so that they can be used by all scenes without having to be recomputed each time.

The next steps are conducted as many times as there are scenes. For the sake of simplicity, the procedure for only one scene is shown.

First the *searchTemplates_A* function is called. This function is intended to find instances of the various models in the scene, if any. For this purpose, the KeyPoints of the scene are taken and the possible Matches between them and the Keypoints of the various models are calculated. In order to obtain these Matches, the FLANN (Fast Library for Approximate Nearest Neighbors) Matcher is used, known to be more efficient in the case of large datasets and for high-dimensional features, along with an approximate KD-tree indexing technique, which allows to organize the descriptors in such a way that it is easier to match them. Of all these Matches, the ambiguous and incorrect ones are eliminated. This is done by calculating the distance ratio between the two closest Matches of a considered KeyPoint and keeping only the Matches that have this value below a threshold.

Once the right Matches for each model are found, the *findPerimetersTemplates* function is called, which compute for each model the perimeter of the instance in the scene. This function uses the RANSAC algorithm to obtain the estimation of the position of the model. RANSAC (Random Sample Consensus) is an algorithm capable of discarding outliers and able to make a robust and correct estimate even with noisy data. It exploits a Homography as a parametric model and the reprojection error as a loss function. Homography is a 3x3 matrix that maps points expressed in homogeneous coordinates from a 3D to a 2D space.

Once the possible corners of the shape of the model within the scene have been found, two checks are conducted to discard any incorrect findings. The first check conducted controls the inclination of the Bounding Box found. Since the program aims to find rectangular boxes of cereals placed on shelves, this check verifies that the shape of the rectangles found by RANSAC is not too inclined and crippled and that the inclination falls within a certain limit.

After this step, the color of the delimited area of the corners found is checked. The average color is calculated as for the models, that is, discarding the first third of the box. The difference between the average color found and that of the analyzed model must be less than a certain threshold. If the two controls are successful, the perimeter of the Bounding Box is calculated and added to a list so that it can be used later.

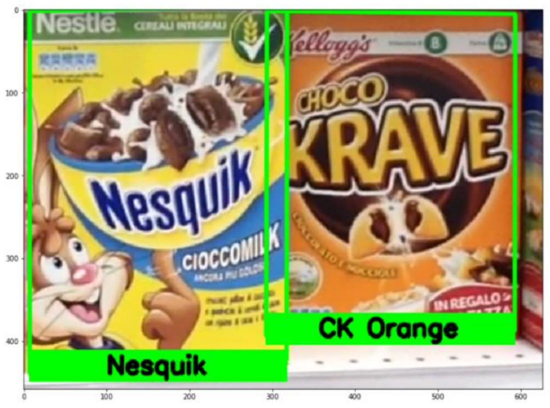
Once all the possible perimeters of the model instances within the scene have been found, the *drawResults* function is called, which has the task of printing them on the image. Since all cereal boxes are similar in size, a check is made on the size of each perimeter before printing it. If the size of a single perimeter differs significantly from the median of all the perimeters found in the scene, it is not considered.

If the perimeter passes the control, a green rectangle of the indicated size is printed on the image in the position calculated, to indicate where a certain model is found within the scene, then the name of the model is displayed under this Bounding Box. Once all the boxes are drawn the *printResults* function is called, which prints the occurrences of the various models within the scene, the height and width of each box and its center, obtained with the *getCentroid* function.

## 2.2 - Results Step A

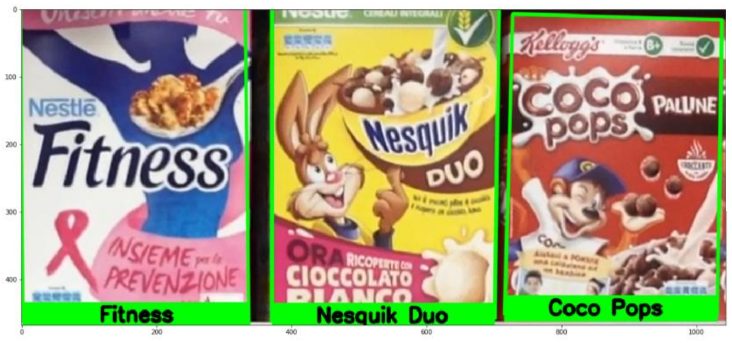
The models used in this step are the number 0, 1, 11, 19, 24, 25 and 26, respectively Nesquik, Choco Krave Blue, Choco Krave Orange, Country Crisp, Fitness, Coco Pops and Nesquik Duo.

The scenes analyzed are e1, e2, e3, e4 and e5. The results are visible in the figures below.



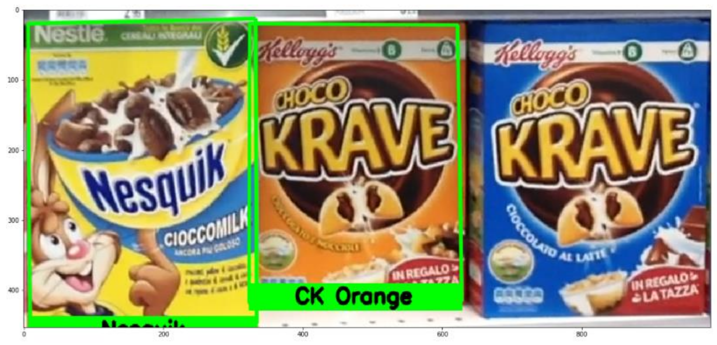
```
Product 0 - 1 instance found:

        Instance 1: {position: (162,215), width: 310px, heigh: 439px}

Product 11 - 1 instance found:

        Instance 1: {position: (442,197), width: 299px, heigh: 393px}
```

Figure 2 – Scene e1 recognition



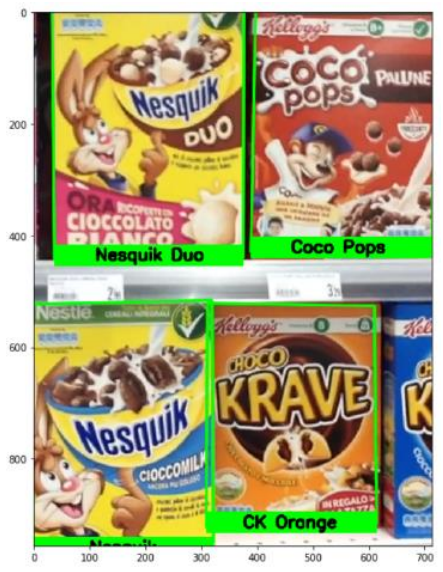
```
Product 24 - 1 instance found:

        Instance 1: {position: (167,219), width: 340px, heigh: 490px}

Product 25 - 1 instance found:

        Instance 1: {position: (878,232), width: 303px, heigh: 445px}

Product 26 - 1 instance found:

        Instance 1: {position: (538,218), width: 330px, heigh: 490px}
```

Figure 3 – Scene e2 recognition



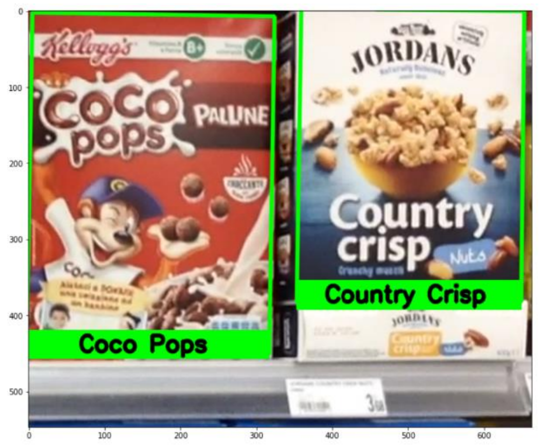
```
Product 0 - 1 instance found:

        Instance 1: {position: (170,236), width: 327px, heigh: 445px}

Product 11 - 1 instance found:

        Instance 1: {position: (474,218), width: 304px, heigh: 398px}
```

Figure 4 – Scene e3 recognition



```
Product 0 - 1 instance found:

        Instance 1: {position: (158,739), width: 327px, heigh: 445px}

Product 11 - 1 instance found:

        Instance 1: {position: (461,721), width: 304px, heigh: 402px}

Product 25 - 1 instance found:

        Instance 1: {position: (555,208), width: 316px, heigh: 448px}

Product 26 - 1 instance found:

        Instance 1: {position: (205,195), width: 343px, heigh: 497px}
```

Figure 5 – Scene e4 recognition



```
Product 19 - 1 instance found:

        Instance 1: {position: (503,190), width: 290px, heigh: 386px}

Product 25 - 1 instance found:

        Instance 1: {position: (161,227), width: 312px, heigh: 444px}
```

Figure 6 – Scene e5 recognition

The Figure 4 shows that the program fails to recognize both instances of Choco Krave cereal boxes. This is because this implementation of the local invariant feature paradigm can only recognize an instance of a model within the scene. In fact, the two cereal boxes are identical in graphics except for the color but the SIFT detector is color invariant since it converts the images to grayscale before detecting the KeyPoints. This error will then be solved in Step B, where multiple instances of the same model are detected.

## 3 - STEP B – MULTIPLE INTANCE DETECTION

The goal of Step B is to detect a single or multiple instances of the products within the images. The images of the models are the same as in Step A, while the images depicting the shelves contain in some cases several boxes of cereals of the same type. The system must now be able to recognize and detect the location of all instances of the models.

This task is solved using the local invariant feature Paradigm together with GHT (Generalized Hough Transform). The steps taken to obtain the results are visible in Figure 7.

All the steps listed are performed for each given scene.

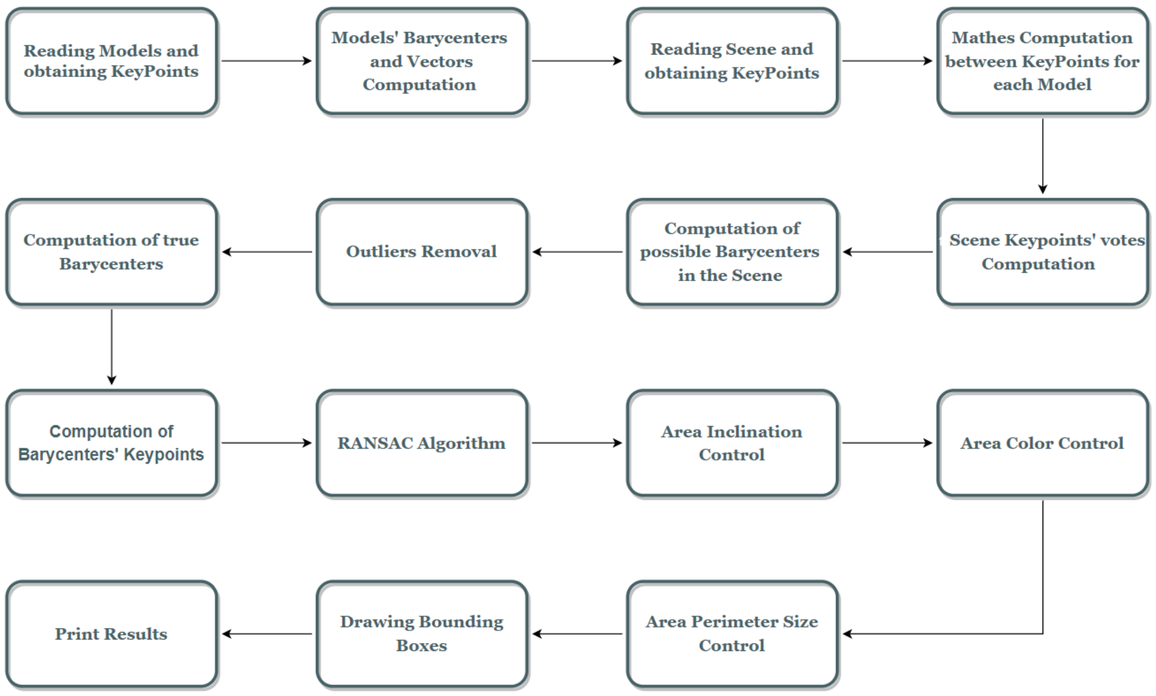

Reading Models and obtaining KeyPoints
Models' Barycenters and Vectors Computation
Reading Scene and obtaining KeyPoints
Mathes Computation between KeyPoints for each Model
Scene Keypoints' votes Computation
Computation of possible Barycenters in the Scene
Outliers Removal
Computation of true Barycenters
Computation of Barycenters' Keypoints
RANSAC Algorithm
Area Inclination Control
Area Color Control
Area Perimeter Size Control
Drawing Bounding Boxes
Print Results

Figure 7 – Step B Diagram

## 3.1 - Implementation Step B

As in Step A, all the images of the models and scenes are firstly read and converted from BGR to RBG. The latter are also read in grayscale to be used by GHT. Then the average colors of the models are calculated as in the previous step.

Once all this information has been obtained, the system analyzes one model at a time, extracts the Keypoints with the *getKeypoints* function and, through the *computeBarycenter* and *computeJoiningVectors* functions, calculates respectively the barycenter of the model, obtained as an arithmetic mean between the coordinates of the Keypoints, and the vectors that join each individual Keypoint to the barycenter found.

The next steps are conducted as many times as there are scenes. For the sake of simplicity, the procedure for only one scene is shown.

What differs this step from the previous one lies in the possibility that there are several equal boxes on the shelf and therefore all instances must be recognized. To obtain the Bounding Boxes of all the models in the scene, the *generalizedHoughTransform* function is called. This function applies GHT. This Hough transformation, being generalized, rather than relying on the classical R-Table, exploits the joining vectors calculated at the previous step for each model.

This function calculates the Matches between the Keypoints of each model and those of the image, using the FLANN Matcher, and selects only the correct ones by analyzing the distance ratio between the two nearest Matches.

At this point it invokes the *voting* function. This has the task of computing for each Keypoint of the scene, matched with a Keypoint of the model, its vote for the position of the barycenter within the scene. It is in this step that the previously calculated joining vectors are exploited, they are appropriately scaled, rotated and added to the coordinates of the point to obtain its vote. All the votes are stored in an Accumulator array, with the same size as the input image.

Two filter are applied on this array to obtain the barycenters of the models. The first filter is applied by the *outliersFilter* function. This filter removes points that are located far from the areas of the image where most of the votes are concentrated. They are therefore outliers and the result of an incorrect KeyPoint's vote and should not be considered. The second filter is applied by the *distanceFilter* function, which groups all the barycenters found, which fall within a certain range of given amplitude, in a single one, calculated as their average.

Once the list has been filtered and the final elements have been obtained, the system calculates which KeyPoints of the scene refer to a certain barycenter and therefore to a certain instance of a model. The *getCorrectKeypoints* function returns for each barycenter found the set of associated scene KeyPoints.

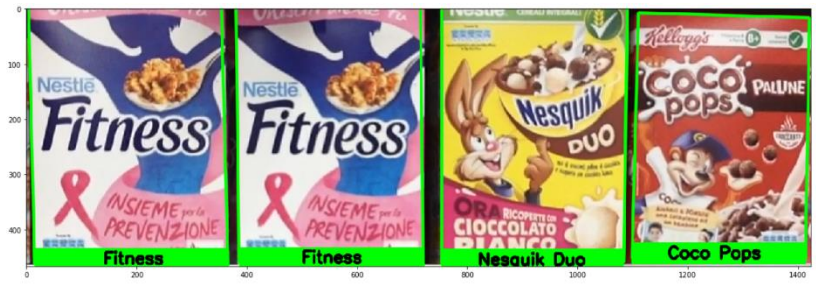
Once the various groups of KeyPoints have been calculated, the procedure is identical to Step A. So, the *findPerimetersTemplates* function is invoked for each group, which calculates the perimeters of the instances of the various models that will then be filtered according to inclination, color and size and only those that pass all the tests will be printed on the final image of the scene.

## 3.2 - Results Step B

The models used in this step are the same as before.

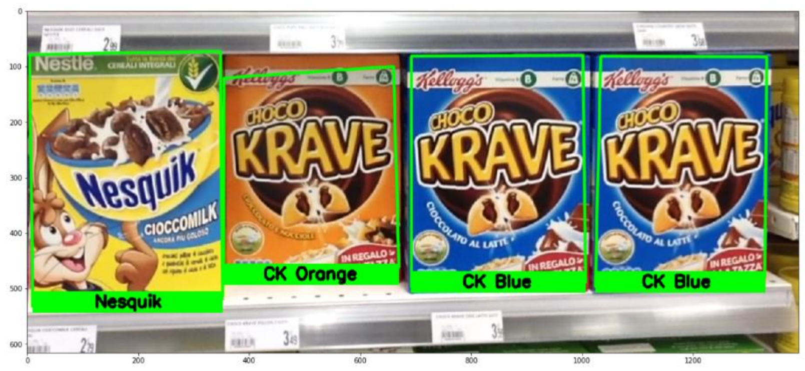
The scenes analyzed are m1, m2, m3, m4 and m5. The results are visible in the figures below.

Figure 13 shows that this method can find both instances of the Choco Krave boxes within the e3 scene, solving the previously exposed error.



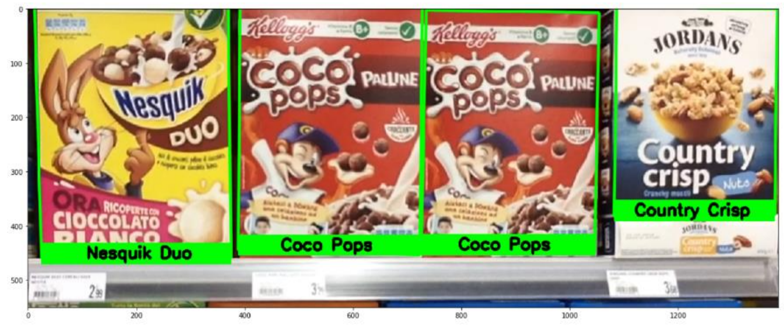
```
Product 24 - 2 instance found:

        Instance 1: {position: (184,214), width: 363px, heigh: 503px}
        Instance 2: {position: (550,219), width: 340px, heigh: 488px}

Product 25 - 1 instance found:

        Instance 1: {position: (1261,232), width: 303px, heigh: 446px}

Product 26 - 2 instance found:

        Instance 1: {position: (921,218), width: 330px, heigh: 490px}
        Instance 2: {position: (921,218), width: 330px, heigh: 491px}
```

Figure 8 – Scene m1 recognition



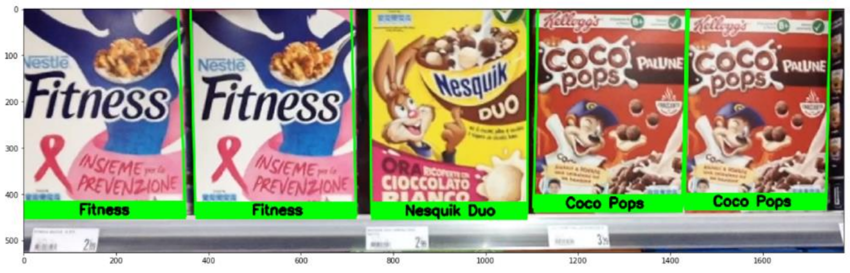
```
Product 0 - 1 instance found:

        Instance 1: {position: (178,300), width: 348px, heigh: 454px}

Product 1 - 2 instance found:

        Instance 1: {position: (1180,285), width: 296px, heigh: 415px}
        Instance 2: {position: (848,287), width: 314px, heigh: 415px}

Product 11 - 1 instance found:

        Instance 1: {position: (509,291), width: 315px, heigh: 362px}
```

Figure 9 – Scene m2 recognition



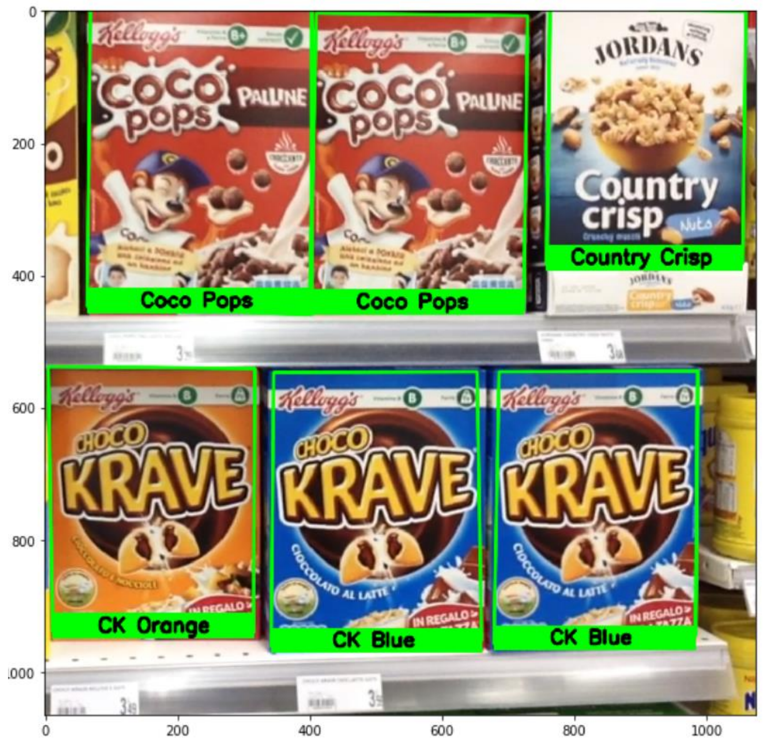
```
Product 19 - 1 instance found:

        Instance 1: {position: (1233,189), width: 291px, heigh: 386px}

Product 25 - 2 instance found:

        Instance 1: {position: (890,226), width: 311px, heigh: 442px}
        Instance 2: {position: (558,221), width: 331px, heigh: 455px}

Product 26 - 1 instance found:

        Instance 1: {position: (195,206), width: 364px, heigh: 509px}
```

Figure 10 – Scene m3 recognition



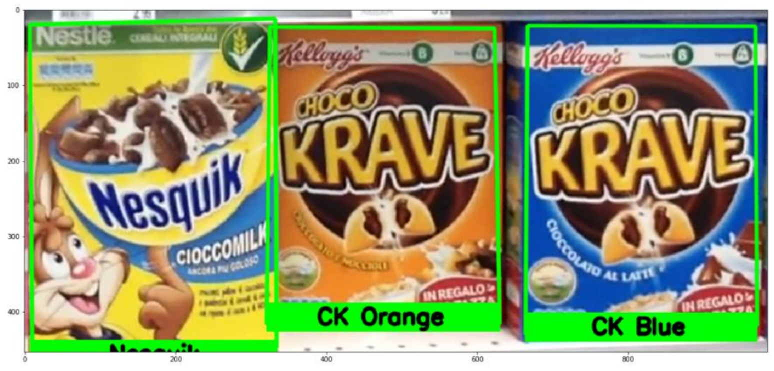
```
Product 24 - 2 instance found:

        Instance 1: {position: (161,193), width: 376px, heigh: 505px}
        Instance 2: {position: (540,196), width: 360px, heigh: 500px}

Product 25 - 2 instance found:

        Instance 1: {position: (1588,211), width: 297px, heigh: 434px}
        Instance 2: {position: (1271,209), width: 316px, heigh: 451px}

Product 26 - 1 instance found:

        Instance 1: {position: (921,195), width: 343px, heigh: 498px}
```

Figure 11 – Scene m4 recognition



```
Product 1 - 2 instance found:

        Instance 1: {position: (500,751), width: 314px, heigh: 414px}
        Instance 2: {position: (833,748), width: 297px, heigh: 411px}

Product 11 - 1 instance found:

        Instance 1: {position: (162,739), width: 310px, heigh: 401px}

Product 19 - 1 instance found:

        Instance 1: {position: (908,190), width: 291px, heigh: 387px}

Product 25 - 2 instance found:

        Instance 1: {position: (235,223), width: 334px, heigh: 457px}
        Instance 2: {position: (566,228), width: 313px, heigh: 446px}
```

Figure 12 – Scene m5 recognition



```
Product 0 - 1 instance found:

        Instance 1: {position: (170,236), width: 327px, heigh: 444px}

Product 1 - 1 instance found:

        Instance 1: {position: (817,222), width: 304px, heigh: 408px}

Product 11 - 1 instance found:

        Instance 1: {position: (474,219), width: 305px, heigh: 393px}
```

Figure 13 – Scene e3 correct recognition