Object detection and Feature Matching

Matteo Bino, Gianluca Caregnato, Federico Meneghetti

1 Contributions and ideas

Initially, together as a group, we discussed the different approaches with which we could tackle the problem. As suggested by the assignment, an obvious method is the creation of a system based on feature matching; however, in addition to this, we decided to try to apply the *Viola&Jones* algorithm to verify whether it was possible to identify objects and not only faces. The aim was to compare the various methods to evaluate their quality and effectiveness. Therefore, the students Bino Matteo and Federico Meneghetti focused mainly on the technique based on feature matching, while the student Gianluca Caregnato explored the functioning of *Viola&Jones*. We clarify that all the problems encountered during the creation of the code were always shared and reworked with all the group members; therefore, even if we divided the tasks in this way, the result produced is to be considered as a global work.

Once the feature matching system was created, we immediately noticed that good results were achieved mainly by identifying the sugar box and, in part, the mustard bottle. So, in an attempt to improve detection accuracy (mainly for the power drill), we applied filters to images and models. The filters used are the following (and in the same order):

- bilateral: $diameter = 5$, $\sigma_{color} = 75$, $\sigma_{space} = 75$,
- CLAHE [1]: $clip\_limit = 3$, $tile\_grid\_size = 8$,
- unsharp mask [2]: $\sigma = 1$, $\alpha = 1.5$;

The final results, as visible in the appropriate section, remained almost unchanged apart from small variations, such in mustard bottle.

As for *Viola&Jones*, despite trying to expand the training dataset by generating synthetic images, unfortunately, the results obtained were not very convincing. Therefore, even considering the time needed to properly train the model, we actually experienced that this is a not very effective technique if you want to identify specific objects and not faces.

The total amount of hours dedicated to this project was about 30+ hours for each member.

2 Code compiling and running instructions

By opening a terminal inside the main folder (the one which contains the files **build.sh** and **run.sh**), the project can be compiled by executing the command **./build.sh**, which uses CMake to build all the dependencies into the newly generated **build** folder. Then, by executing **run.sh** one can effectively execute the program that produces an **output** folder. If you want to test the program using other dataset's directories or output directories, it is enough to modify the two paths specified inside **run.sh** file (default ones are **../dataset** and **../output**). If you want to generate the dataset used for the training of the models, you can do it running **./runGenerator.sh**. The models are already trained so it is not mandatory.

3 Overview of the Code

3.1 Dataset and Utils functions

Dataset is the class used to load, index and share the images of the datasets. For each object type (sugar box, mustard bottle, power drill) one Dataset object is created, and in this class are stored the test images -and models- filenames. *Utils* is a namespace used to group some useful functions to ease every task, such as loading images, labels, splitting paths and implementing functions to measure object detection accuracy and to log results.

[1] Contrast Limited Adaptive Histogram Equalization: https://en.wikipedia.org/wiki/Adaptive_histogram_equalization#Contrast_Limited_AHE

[2] USM (unsharp mask) https://en.wikipedia.org/wiki/Unsharp_masking

3.2 ObjectDetector

ObjectDetector is an abstract class that generally defines the ability of a class to identify an object from an image, returning a Label. A Label specifies the type of the returned object and the corresponding bounding box in the reference image.

Therefore two classes have been defined that, by extending the *ObjectDetector*, try to solve the problem of identifying an object with different approaches. Specifically, the implemented classes are *FeaturePipeline* and *Viola&Jones*.

3.2.1 FeaturePipeline

FeaturePipeline overrides the virtual function *detect_objects* by implementing a system based on feature matching. In particular, the main steps performed by the *FeaturePipeline* class are the following.

- First, once the object of interest has been chosen, it is necessary to extract the features of each available model. This is an operation performed in advance, that is, before the invocation of the *detect_objects* function, to allow the pipeline to know the characteristics of the object to be identified.
- We then proceed by extracting the features of the image on which we want to identify the object.
- Now the matches between the features of the models and the target image are extracted, to select the best model, that is, the one that produces the highest number of 'good' matches evaluated through Lowe's test.
- Finally, we proceed using the matches of the best model to identify the bounding box of the object in the image.

Eventually it is possible to preprocess images using some filtering methods implemented in the *ImageFilter* class, which allows to concatenate arbitrary filters in a pipeline.

3.2.2 Viola&Jones

The Viola-Jones algorithm requires a large number of samples to effectively learn a model. To increase the number of training samples, we decided to develop an image generator. The image generator takes an image from the dataset as input, along with the number of images the user wants to generate. It outputs several variations of the input image by applying rotations and changes in lighting conditions. To use Viola-Jones, it is necessary to generate a `.txt` file containing the paths of all the training images, as well as a separate file listing all the test images. Training the models is very slow: with the smallest feature window size 24×24, it took about 2 hours per model, while using a feature window of 64×64 — which gave us much better results — required approximately 2 days.

3.3 Object detection accuracy and output image saving

The labels returned after running ObjectDetection will be evaluated in terms of accuracy. For each object type (sugar box, mustard bottle, power drill), one folder is created and inside it it's produced a CSV file containing the accuracy and mean IoU of every object detection method that we implemented. Additionally, for every object detection method it is created a folder containing all the test images with the relative labels (both true labels in green and predicted labels in red).
In the following page there is a table to sum up every object detector's accuracy and mean IoU, followed by the image results of these detectors.

Note bounding boxes detected by our models are marked in red, while ground truth bounding boxes are in green.

3.4 FeaturePipeline

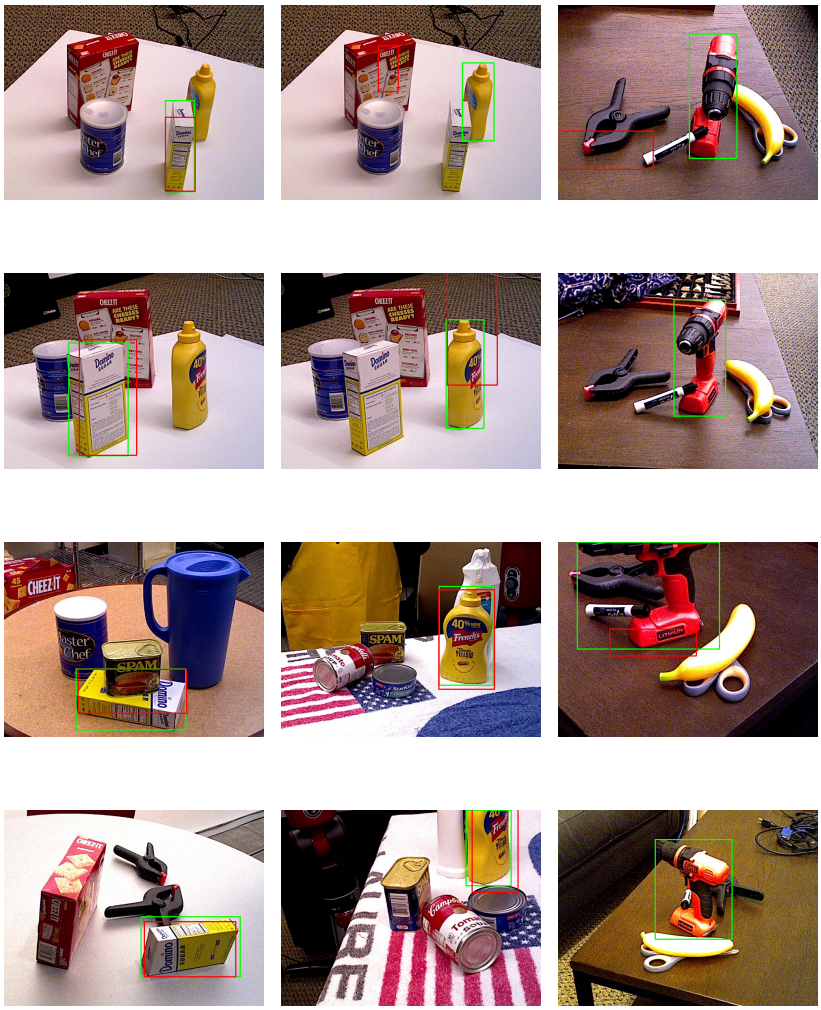
3.4.1 Detection Accuracy

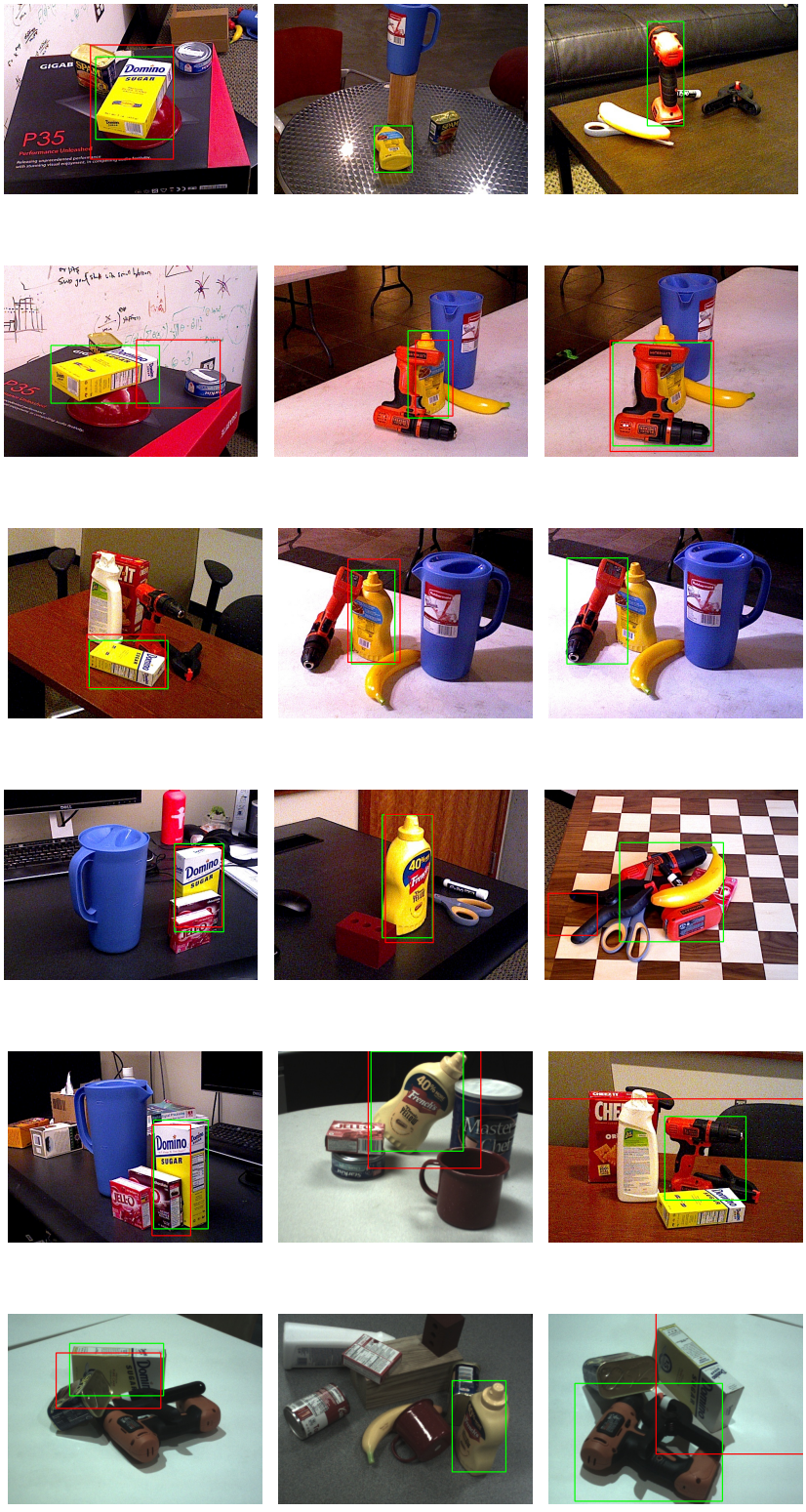
Object Type	Detector and Matcher	Model Filters	Scene Filters	Accuracy	MeanIoU
004_sugar_box	**SIFT-FLANN**			**0.9**	**0.671773**
004_sugar_box	SIFT-FLANN	Bilateral-CLAHE-Unsharp	Bilateral-CLAHE-Unsharp	0.7	0.587006
004_sugar_box	ViolaJones	GaussianBlur	GaussianBlur	0	0.0406089
006_mustard_bottle	SIFT-FLANN			0.5	0.456412
006_mustard_bottle	**SIFT-FLANN**	**Bilateral-CLAHE-Unsharp**	**Bilateral-CLAHE-Unsharp**	**0.5**	**0.465672**
006_mustard_bottle	ViolaJones	GaussianBlur	GaussianBlur	0	0.00498579
035_power_drill	SIFT-FLANN			0.1	0.11703
035_power_drill	SIFT-FLANN	Bilateral-CLAHE-Unsharp	Bilateral-CLAHE-Unsharp	0	0.0681583
035_power_drill	**ViolaJones**	**GaussianBlur**	**GaussianBlur**	**0.3**	**0.391784**

Table 1: Risultati del test

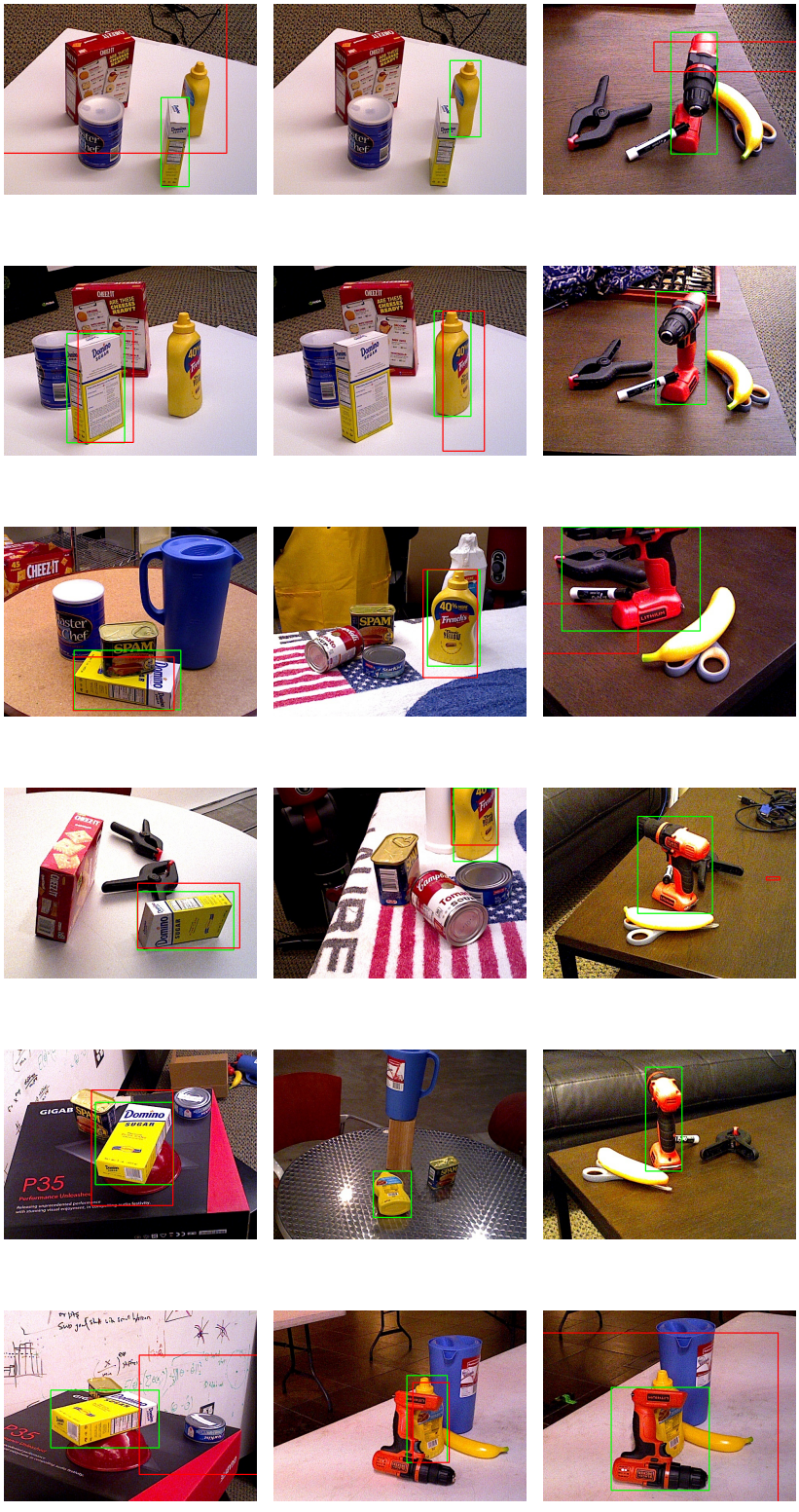
3.4.2 Images

3.4.3 SIFT-FLANN images



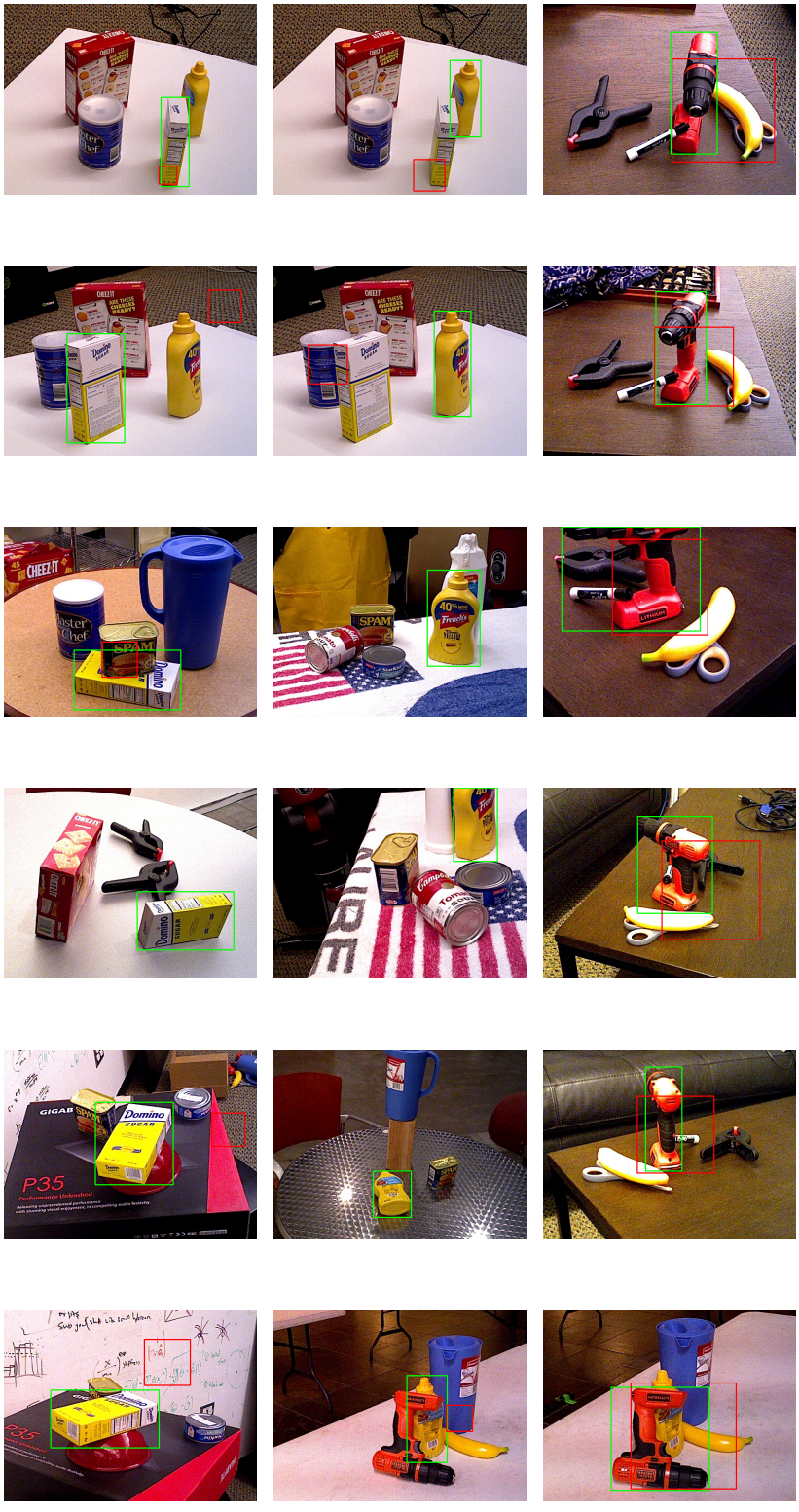

Domino
SUGAR
P35
Performance Unleashed
JELL-O

3.4.4 SIFT-FLANN-Filtered images




Domino
SUGAR
Domino
SUGAR

3.4.5 Viola&Jones images




Domino
SUGAR
Domino
SUGAR