University of Padova

Computer Vision Final Project

Object Recognition with SIFT, SURF and LBP

Student:

Umberto Michieli (1150780)

Instructor:

Pietro Zanuttigh

Abstract

One of the main topics of the Computer Vision is surely the object detection and recognition inside a scene because it is useful in a large number of applications. The algorithms that solve this problem are based on the extraction of some features, like for example corners or edges or blobs, from the object and from the scene (feature detection) and associate them to a distinctive descriptor in order to distinguish a determinate feature among others.
In this report the SIFT, the SURF and the LBP methods are analyzed trying to point out their strengths and weaknesses through many results.

Moreover a few motion detection and estimation tasks have been realized.

June 7, 2017

1 Introduction

The task of object detection and recognition is surely one of the most important topics of Computer Vision and his applications are really many, as for example gesture recognition (see [1]), video tracking (as in [2] and [3]), match moving (see [4]), image stitching (as in [5]), 3D modeling, robotic mapping (as in [6]), navigation and many others.

This report is intended to be an overview of a few algorithms that have been developed in literature in order to solve this task. I implemented all the procedures presented in Matlab exploiting both the VLFeat library (an open source library) and the Computer Vision System Toolbox (developed by Matlab itself). All the scripts, used to obtain all the images presented later on, are attached to this file (note that some scripts take quite a long time to execute depending on the image size -for example the images taken with my phone are quite heavy because less compressed- and on the parameters' choice for accuracy and number of iterations). In this project I considered the well-known algorithms of feature detector and description of SIFT, SURF and LBP.

Furthermore, in order to match my personal interests I have also done some tasks regarding video motion detection and estimation.

The remainder of this paper is organized as follows. In the next section the objectives and a short theoretical recall of the algorithms used are proposed. In the third section all the results obtained are presented and were made some considerations. The fourth section discusses a simple algorithm to stitch images exploiting SIFT. The fifth section provides some results for the motion estimation of videos. The sixth section concludes the paper and discusses some possible expansions of this work.

2 Technical Approach

2.1 Objectives

The main aim of this project is to detect a given object in some scenes and to locate it inside the image or video. To this purposes many algorithms exist and some of them are implemented in the codes attached to this report.

2.2 Scenario

In order to identify some objects in the scenes we generally want to build a descriptor associated to a detected feature (either a corner or a edge or a line or a blob or other) which should be able to distinguish that point among others. This can be achieved, ideally, if the descriptor is geometrically invariant, robust to occlusion, robust to noise, robust to blur, robust to illumination changes, distinctive, stable and accurate. As many times in these types of problems we cannot achieve all this desirable properties in a single shot but we need to achieve a trade-off among them.

Many algorithms have been proposed in the literature so far like SIFT, PCA-SIFT, SURF, GLOH, Shape Context, LBP, BRIEF, ORB, MSER just to name a few.

In this report I tried to review three of them, namely the SIFT, the SURF and the LBP. In the following part of this section I'm going to briefly recall the main ideas of these three algorithms, a complete description would take surely too long for the objective of this report.

1. **SIFT (Scale Invariant Feature Tranform):** is one of the oldest method (proposed in 1999 [7]) but still probably the most reliable. It provides both the feature detection and the feature description. As the name says, it offers a scale-invariant result by looking for the keypoints (i. e. features) on the whole scale space. The procedure can be subdivided into four steps: the *scale-space extrema detection* finds minima and maxima in the image through the Difference of Gaussians (DoG) of eight neighbors on the scale considered and nine points on the previous and on the next scale levels; the *keypoint localization* tries to locate the maxima with the Taylor series approximation of the DoG in the neighborhood of the point considered taking only the more stable keypoints by applying a threshold on the amplitude of the gradient; the *orientation assignment* associates an orientation based

on the gradient to each keypoint making the descriptor invariant with respect to the scale and the rotation; finally the *keypoint descriptor* is built exploiting the gradient modules and the gradient rotation.

2. **SURF (Speeded Up Robust Feature) [8]:** it tries to speed-up the computations using the integral image, i. e. a sum of all the pixels in the top-left rectangular window of the image with respect to the point considered. It provides both feature detector and descriptor; is partially inspired by the SIFT algorithm but much faster even though less robust as regards illumination and viewpoint changes.

3. **LBP (Local Binary Patterns) [9]:** also this method is very old but very simple and fast. Its main idea, originally, was to divide the image into cells and compare the considered pixel to each of its eight neighbors, express the differences of the intensity between the surrounding pixels and the point itself with an eight-digit binary number; then compute the histogram over the cell of the frequency of each number occurring and concatenate the histograms of all the cells in order to obtain a feature vector. In particular it is invariant to translation but not invariant for rotation and scale. Now some extensions have been made in order to make it more robust and suitable for various applications. Differently from the previous two, it only provides feature description but not feature detection.

2.3 Complications found

Sometimes it was not totally trivial how to interpret the outputs of some functions and the documentation was not exhaustive. For example I would have liked to implement LBP from zero rather than to use the built-in function of Matlab but this revealed to be a long work to do and probably not worth it, because the expected results were much worse than the other feature descriptors hence just a few considerations of LBP were analyzed in this report.

3 Results

3.1 *Baseline* Approach

In this section I briefly recall the matlab functions and the results of the baseline algorithm, already discussed during the sixth laboratory's report. The main script is `mainSIFT.m` and makes use of many others scripts:

- `getAffine(obj, scene, T, thresh, num_matches, set_card, obj_num)`: *obj* is the image of the object we want to detect in the image of the *scene*, *T* is the threshold of the RANSAC algorithm, *thresh* is the matching threshold (typically set around 0.65), *num_matches* is the minimum number of total matches we want to have in order to proceed in the computation of the affine transformation, *set_card* is the minimum cardinality of the largest compatible set must be (i. e. we discard the matches for which the largest compatible set is not very numerous, because it could lead to non-stable results) and *obj_num* is simply an identification number of the current object. The function computes the matches between the SIFT features and then estimates the location of the object in the scene providing in output the affine transformation (i. e. the matrix $\mathbf{M}$ and the vector $\mathbf{t}$). In order to avoid false matches I used a simplified RANSAC algorithm with 700 iterations searching for groups of features with the largest consistent set (i. e. with similar affine transform using a threshold T on the difference between the projections of the keypoint and their locations: $|\Delta u_{max}| + |\Delta v_{max}| < T$, where u and v represent the coordinates of the *scene* image).

- `im_add(obj, M, t, colour)`: this function computes the extremal points on the scene of the object *obj* given their matrix $\mathbf{M}$ and vector $\mathbf{t}$. The *colour* must be a number between 1 and 4, it selects the colour of the polygon to be superimposed to the scene in order to properly identify the object.

- `drawPolygon(p1, p2, p3, p4, colour)`: draws a polygon in the current figure given four points *p1*, *p2*, *p3*, *p4* in a *consecutive* order; it uses `drawLine` to plot the lines.

- `drawLine(p1, p2, colour)`: a simple code that plots a line between the points p1 and

p2 with the specified colour (*colour* must be one element of {1, 2, 3, 4}).

- `read_img_extension(folder_name, common_matrix, extension)`: reads all the images whose names begin with *common_matrix* in the folder *folder_name* in the *extension* format. Note that the folder must be in the same directory of the script.

The first dataset is the simplest among the ones provided. Here the code is able to perfectly recognize all the three objects and to correctly identify all of them in the scene (Figure 1).

The second dataset works perfectly only in the first and in the last scene, while in the two middle scenes the algorithm recognize only partially the scene or detect wrong matches (in order to avoid this undesirable effect one can think of properly tuning the matching threshold but manually trying I have not found a thresholds' choice good for every scene in the dataset). See Figures 2, 3, 4 and 5.

The third dataset is much simpler than the previous one and the algorithm is able to correctly detect the objects in both the proposed scenes (Figures 6 and 7).

The fourth dataset is by far the most challenging one with many wrong matches due to illumination changes and strong reflections (notice the large number of illumination reflections on the cars' surfaces because the objects are quite polished). Here the baseline approach does not provide robust and repeatable results: either the algorithm almost detect the location but not the actual contours of the objects or it is not able to detect the objects (again, notice that this, in principle, could be manually adjusted by the value of the thresholds but the number of wrong matches was too high and with this approach no good solution have been found). See Figures 8, 9 and 10.

The first extension of the laboratory regards the acquisition of some images from my photo camera; I have taken the following images with an iPhone 6S which, in particular, does not allow the user to maintain a fixed illumination profile, hence the resulting images can unfortunately be more or less brighter depending of the object considered. The following results take a while to be generated with the provided script because the images are quite heavy.

For example I considered three different datasets even though here I am going to report the last two of them, basically because the first scenario was not properly selected: i. e. the objects considered were quite uniform with a few recognizable points or with polished surface and the scenes were affected by illumination artifacts.

Figure 1: Dataset 1

Figure 2: Dataset 2, scene 1

Figure 3: Dataset 2, scene 3

Figure 4: Dataset 2, scene 4

Figure 5: Dataset 2, scene 5

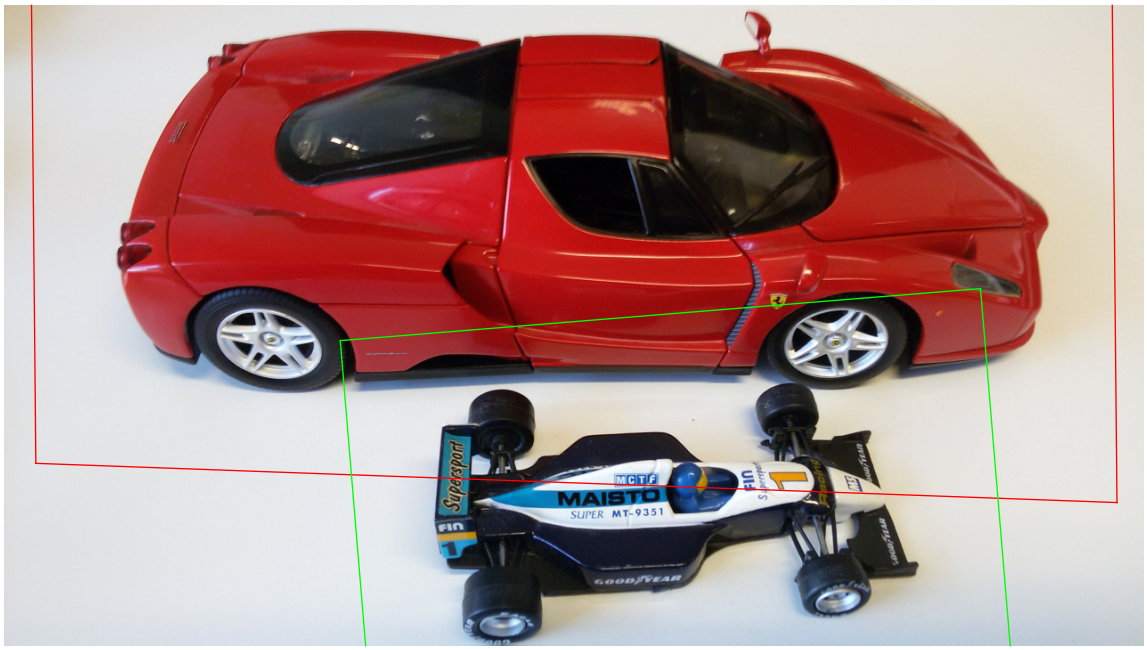

Figure 6: Dataset 3, scene 1

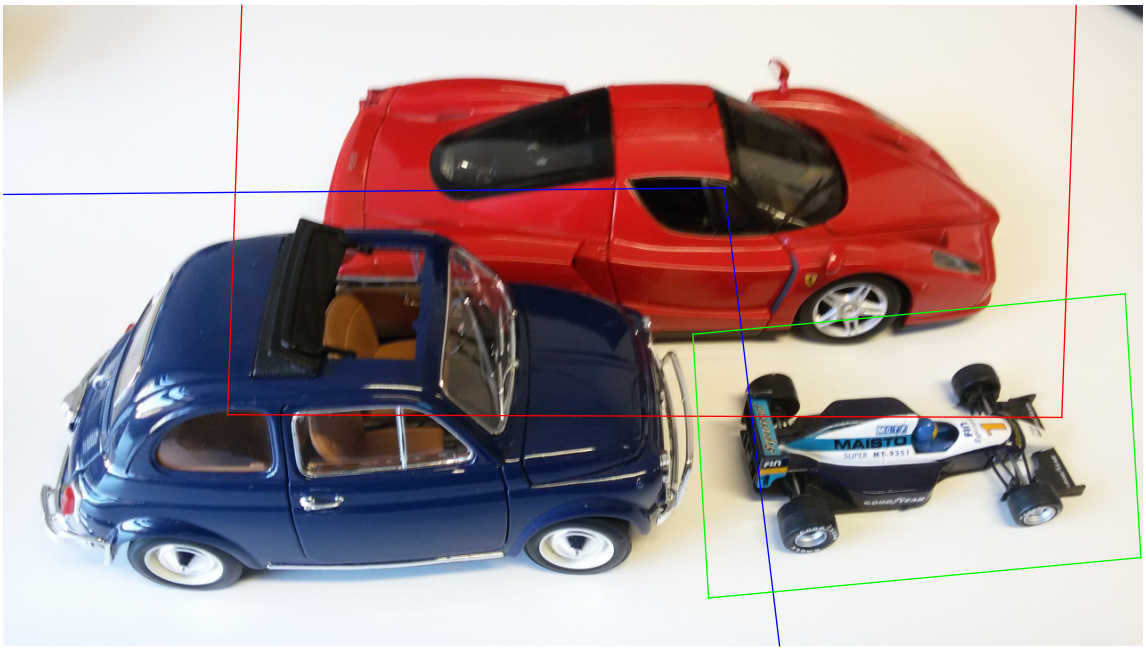

Figure 7: Dataset 3, scene 2

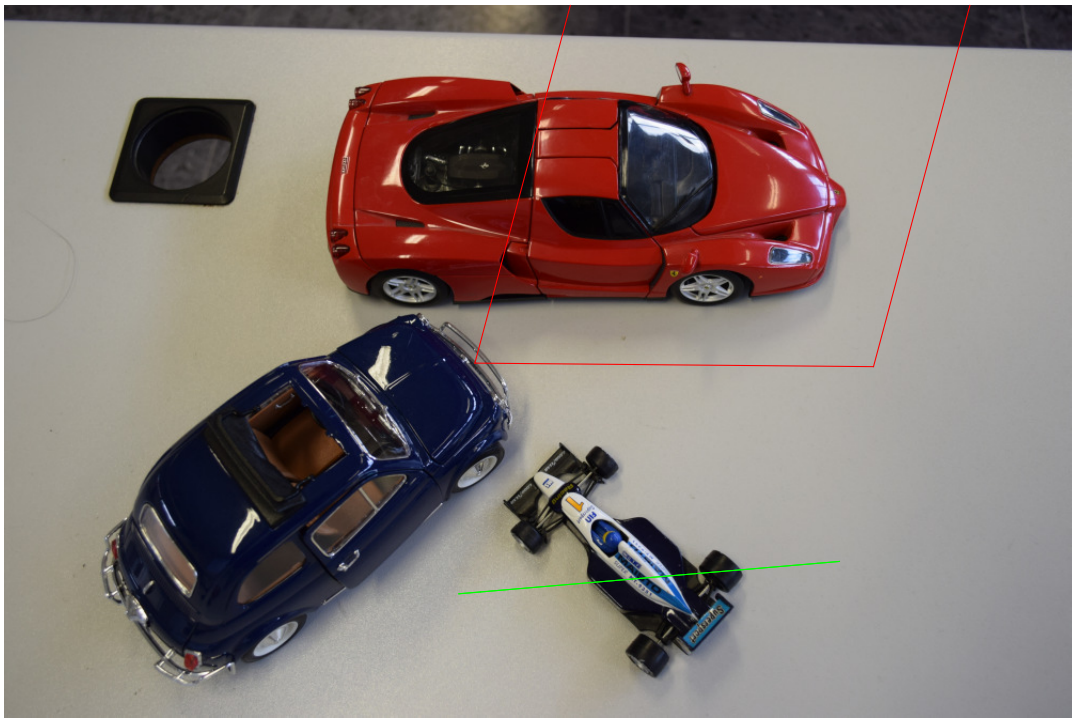

Figure 8: Dataset 4, scene 1

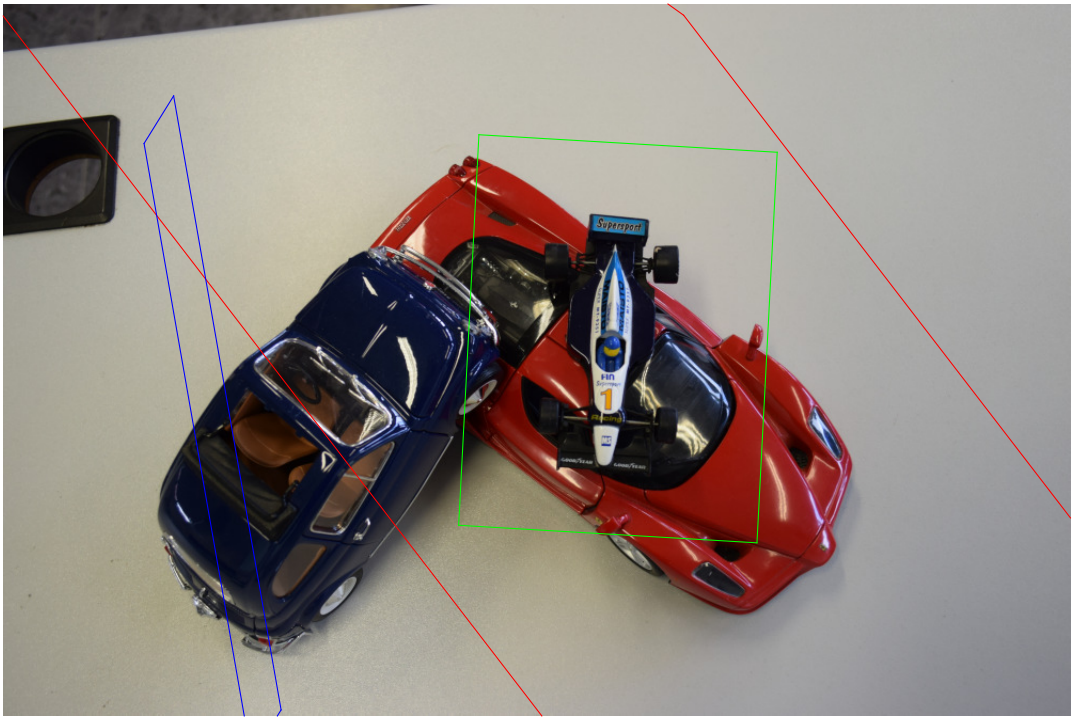

Figure 9: Dataset 4, scene 2

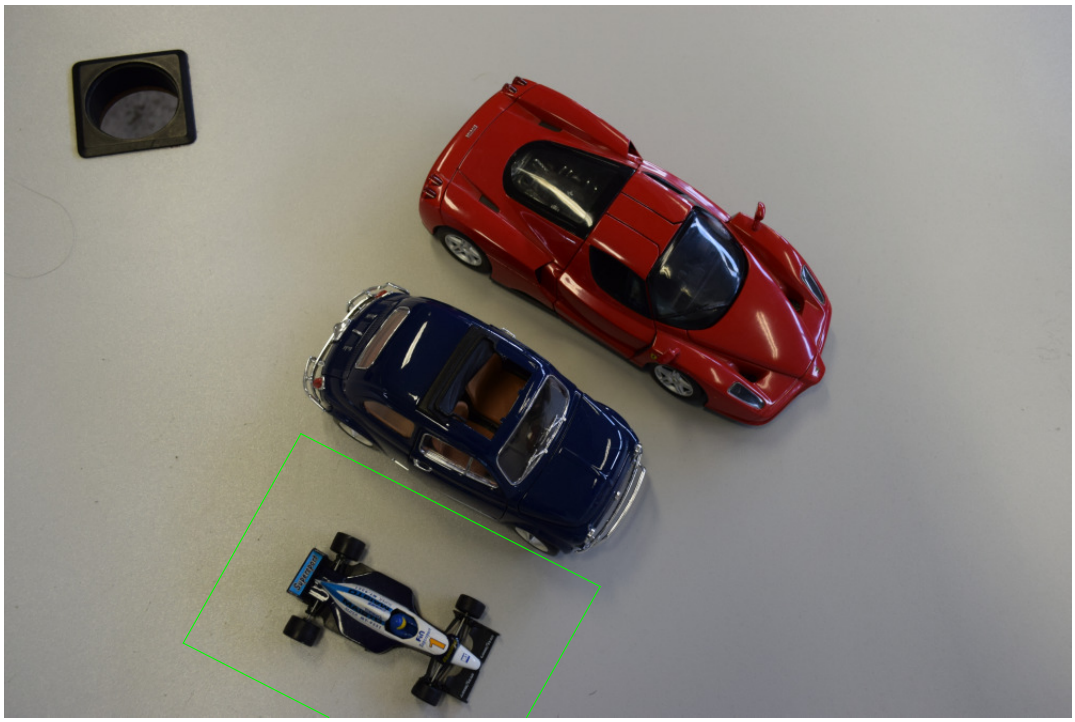

Figure 10: Dataset 4, scene 3

My second dataset is perfectly working with the basic approach implemented since all the objects are recognized even in situations of partial occlusion and superimposition. The parameters here imposed are the same as the ones used previously (matching threshold of 0.65 and RANSAC threshold of 3). See the Figures 11, 12, 13 and 14; note that the algorithm was capable to detect also the pack of paper towels which has a glossy surface because I tried to maintain the same illumination between the photos.

The third dataset proposed (Figures 15, 16, 17 and 18) was set in order to see how the algorithm behave when there are similar objects in the same scene (in particular the rubber eraser and the chewing gum almost have same shape and dimension and similar colors). I found out that the algorithm does not have any particular problem and it is able to recognize all the objects even when they are partially occluded. Note that the colors of the objects does not influence the solution because the computations are done on the respective grayscale images. The only imprecision is in the first scene where the gum is not detected because was collocated in an unusual position and presents some reflections on its surface. On the other hand, note the high precision in all the other scenes and in particular to the second and the third one which were not trivial.

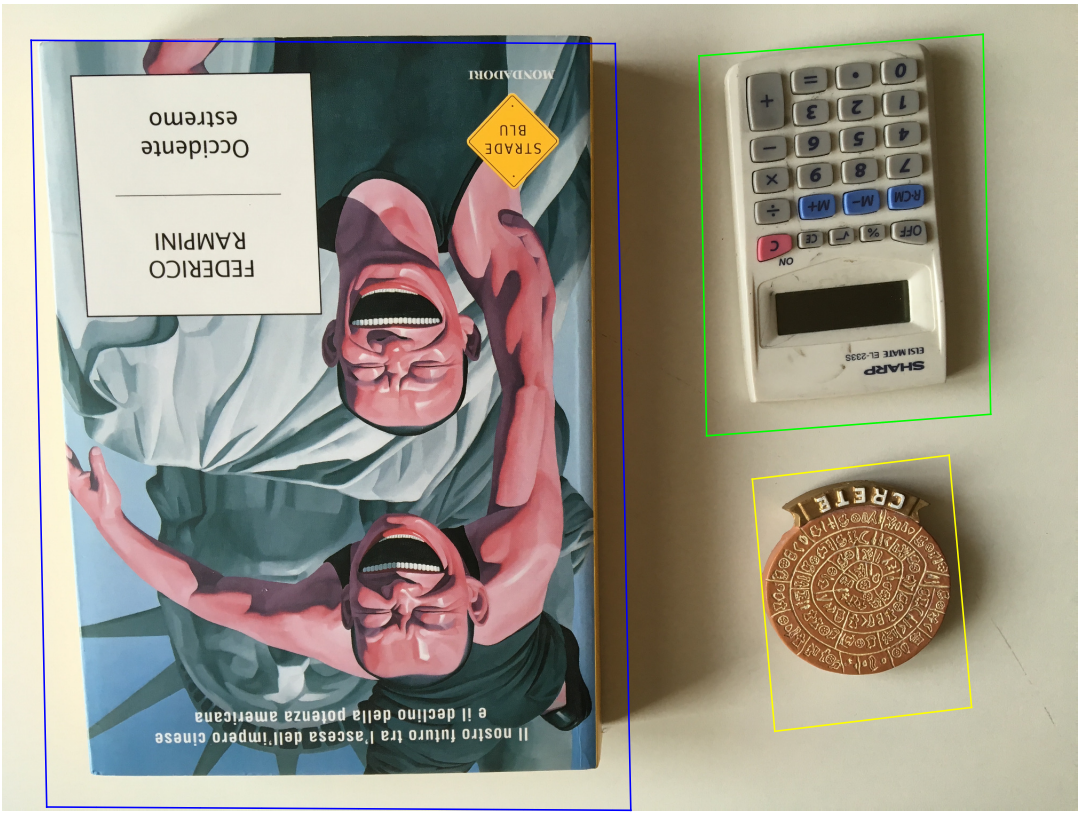

Figure 11: My dataset 2, scene 1

Figure 12: My dataset 2, scene 2

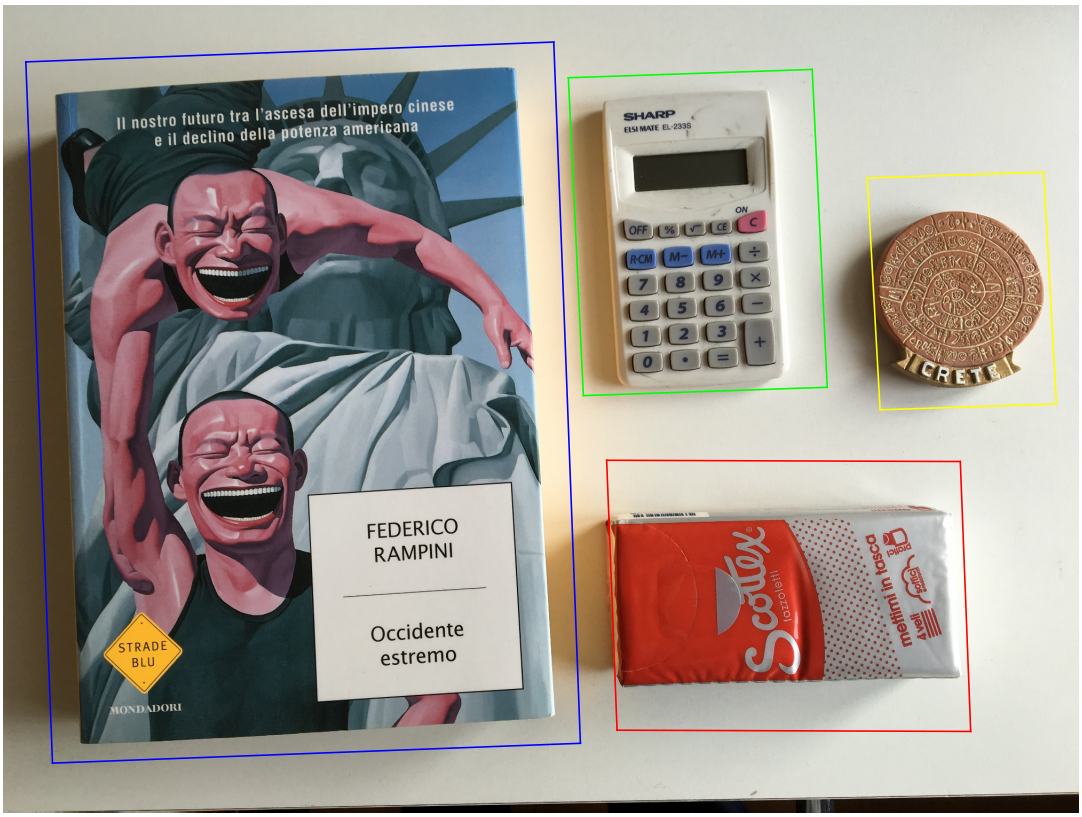

Figure 13: My dataset 2, scene 3

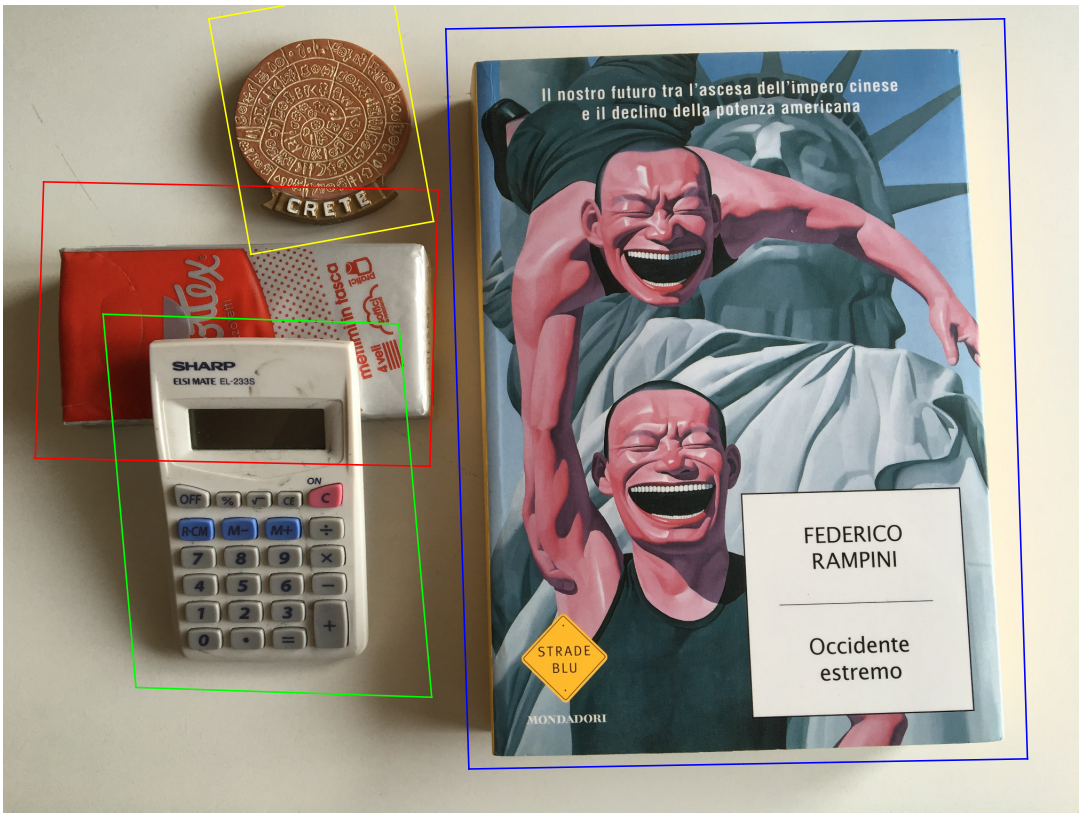

Figure 14: My dataset 2, scene 4

Figure 15: My dataset 3, scene 1

From all these examples above we can conclude that the SIFT method performs very well except in extreme situations or in presence of illumination changes. This sentence is even more important if we think that the SIFT methods was proposed by David Lowe in 1999 [7] and the idea beyond is quite simple even if not very fast (as we can see performing the computations).

3.2 SVD-based methods

Moved by the not-always super precise results one can think of exploiting the Singular Value Decomposition (SVD) in order to obtain a better estimation of the affine transform.
I identified three slightly different ways of using the SVD which are briefly discussed in the following (are just small variations of the algorithm implemented by the script `getAffine` already discussed above, now I used 1000 iterations of the RANSAC algorithm).

1. `getAffineSVD1:` this algorithm picks three random points and computes the affine transformation as done in `getAffine`, the difference is that the matrix $\mathbf{M}$ and the vector $\mathbf{t}$ are improved afterwards by computing the SVD of all the points in the compatible set

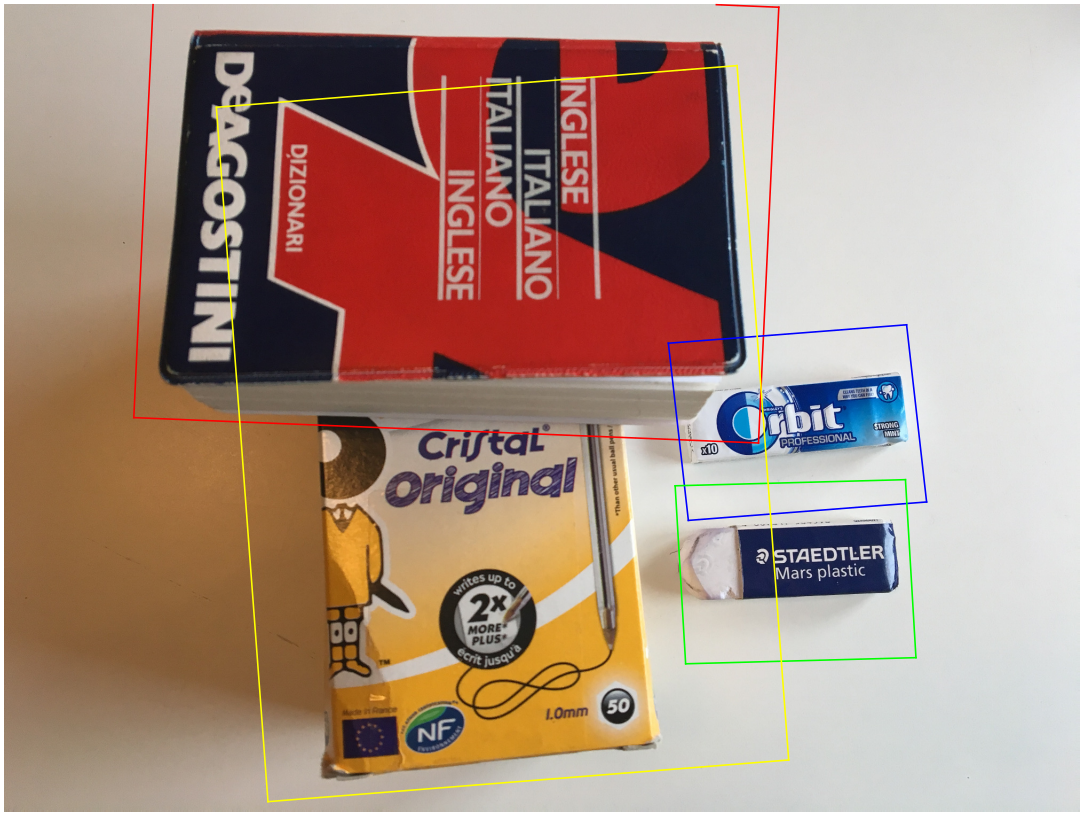

Figure 16: My dataset 3, scene 2

(remember that SVD provides a regularization of the over-determined problem in least square sense).

2. `getAffineSVD2:` the second proposal picks L random points and resolves using SVD in order to obtain M and t; then the affine transformation with the largest compatible set is considered. Note that here L must be not too large because we are regularizing the problem in an l^2 sense, hence the larger errors have a greater impact on the result and in this application we really have a lot of false matches. Picking L of about 4, 5 or 6 the algorithm works generally well because the probability of finding no false matches during an iteration is smaller.

3. `getAffineSVD3:` finally this algorithm computes **M** and **t** using SVD on L random point and builds the compatible set of points with this tranformation; then it recomputes **M** and **t** using all the elements belonging to the largest compatible set. Also in this case hold similar notes as in the previous case for the choice of L (between 4 and 6 generally).

The results here obtained (using the main scipt `mainSVD.m`) heavily depend on the input

Figure 17: My dataset 3, scene 3

Figure 18: My dataset 3, scene 4

Figure 19: SVD method 1, dataset 2, scene 1

parameters passed to the methods above, especially on L (good values are around 4, 5 or 6) and the thresholds for the matching (about 0.6) and the RANSAC (about 2 or 3). In the following a comparison of the three methods is proposed using the second dataset provided (because it is the one that sometimes was used to perform well but some others not so much).

Unfortunately the results were not as good as I expected and basically identical to the ones obtained with the baseline approach, because of some other limitations of the procedure: e. g. the simplifications on the RANSAC procedure and the absence of a dynamic way of thresholding. Figures from 19 to 22 regards the first SVD algorithm, Figures from 23 to 26 the second and from 27 to 30 the third one (compare them with Figures from 2 to 5).

In dataset 4, which was the more challenging one, small improvements can be seen only in the second scene using `getAfineSVD2` where the procedure was able to detect both the Formula 1 and the red cars but wrongs the identification of the blue car (see Figure 31 and compare it with Figure 9).

Also in the other datasets proposed above this algorithms perform very similar to the *baseline* approach.

Figure 20: SVD method 1, dataset 2, scene 3

Figure 21: SVD method 1, dataset 2, scene 4

Figure 22: SVD method 1, dataset 2, scene 5

Figure 23: SVD method 2, dataset 2, scene 1

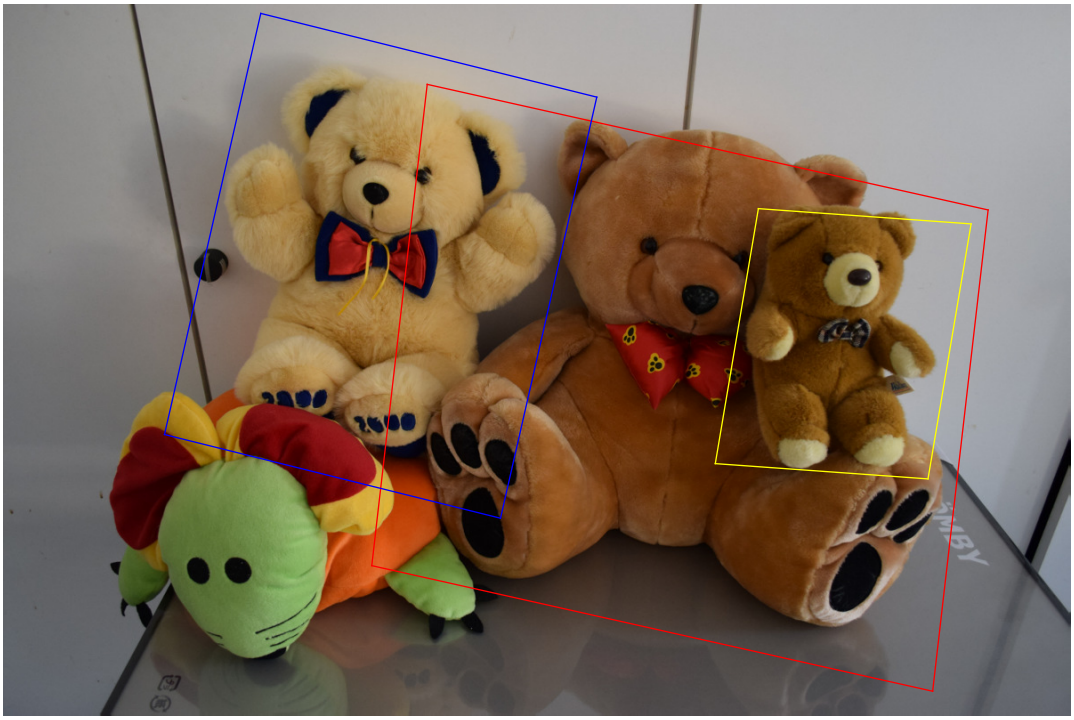

Figure 24: SVD method 2, dataset 2, scene 3

Figure 25: SVD method 2, dataset 2, scene 4

Figure 26: SVD method 2, dataset 2, scene 5

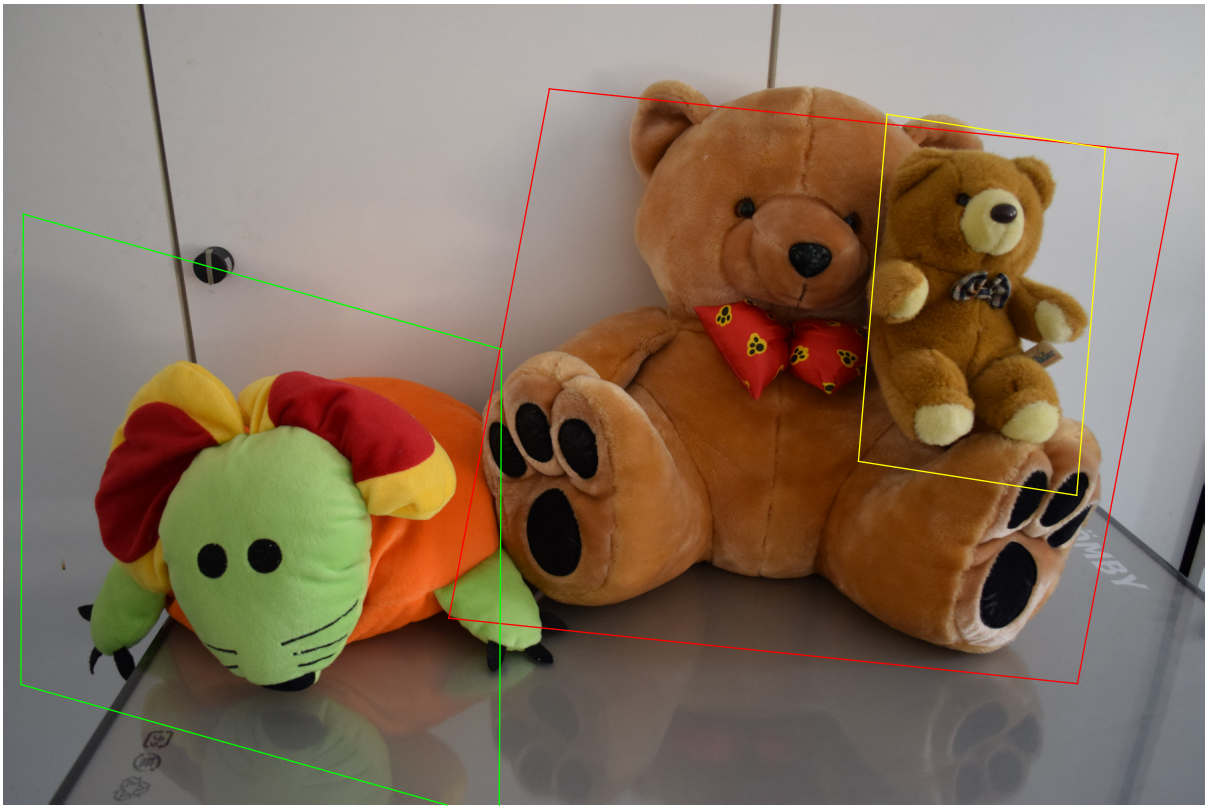

Figure 27: SVD method 3, dataset 2, scene 1

Figure 28: SVD method 3, dataset 2, scene 3

Figure 29: SVD method 3, dataset 2, scene 4

Figure 30: SVD method 3, dataset 2, scene 5

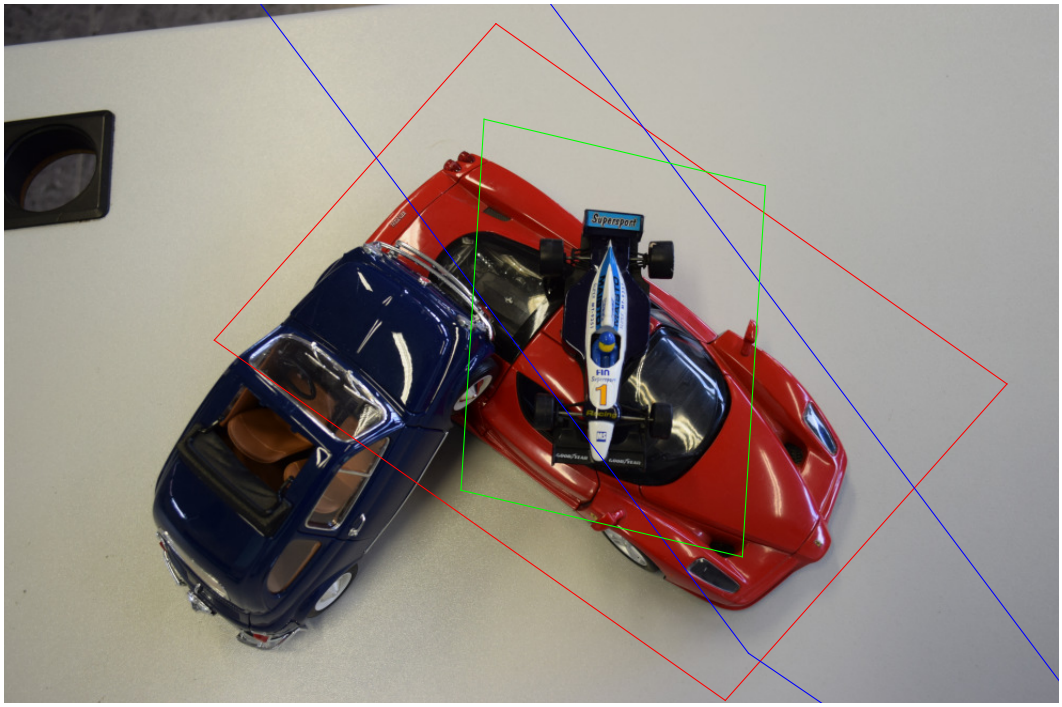

Figure 31: SVD method 2, dataset 4, scene 2

Note that we can think of using a different type of distance norm in the RANSAC algorithm, for example an l^2 norm, instead of the l^1, but no sensible changes were noticed.

3.3 Others Feature Detection and Descriptors

3.3.1 SURF

The SURF method is implemented in the Matlab's Computer Vision System Toolbox. Based on some built-in functions I wrote other two scripts:

- `mainSURF:` is the main scipt to run, it computes the SURF features and descriptors on some of the datasets presented above.
- `getAffineSURF(obj, scene, thresh, T):` it computes the SURF feature and detectors of the image of the object *obj* and the image of the scene *scene*, then matches the corresponding features where *thresh* is a ratio threshold for rejecting ambiguous matches and *T* expresses the distance threshold required for a match. It implicitly assumes an estimation of the geometric transform through MSAC algorithm (a variant of the RANSAC) with a maximum of 5000 number of random trials and 0.1 of maximum distance in pixels that a point can differ from the projection location (here the default value was 1 but for better results the number must be decreased at the expense of quite many additional computation).

In Figure 32 an example of the 50 strongest features extracted through SURF method is reported.

For the sake of comparison I decided to report here the results for object matching obtained on the second (see Figures from 33 to 36) and third dataset (see Figures 37 and 38); some others images can be found in the folder "SURF_results" attached to this file.

Again, note that the results heavily depend on the parameters' choice and in general results similar to the previous methods can be achieved properly tuning them.

Based on the experimental results, it is found that the SIFT generally detects a larger number of features compared to SURF but it is suffered with speed. In fact SURF is much faster (about 2 or 3 times faster than SIFT depending on the input parameters and on the number of iterations) and has quite good performances similar to SIFT even though it is a bit less robust.

Figure 32: Example of feature detection using SURF method on an object of the dataset 2.

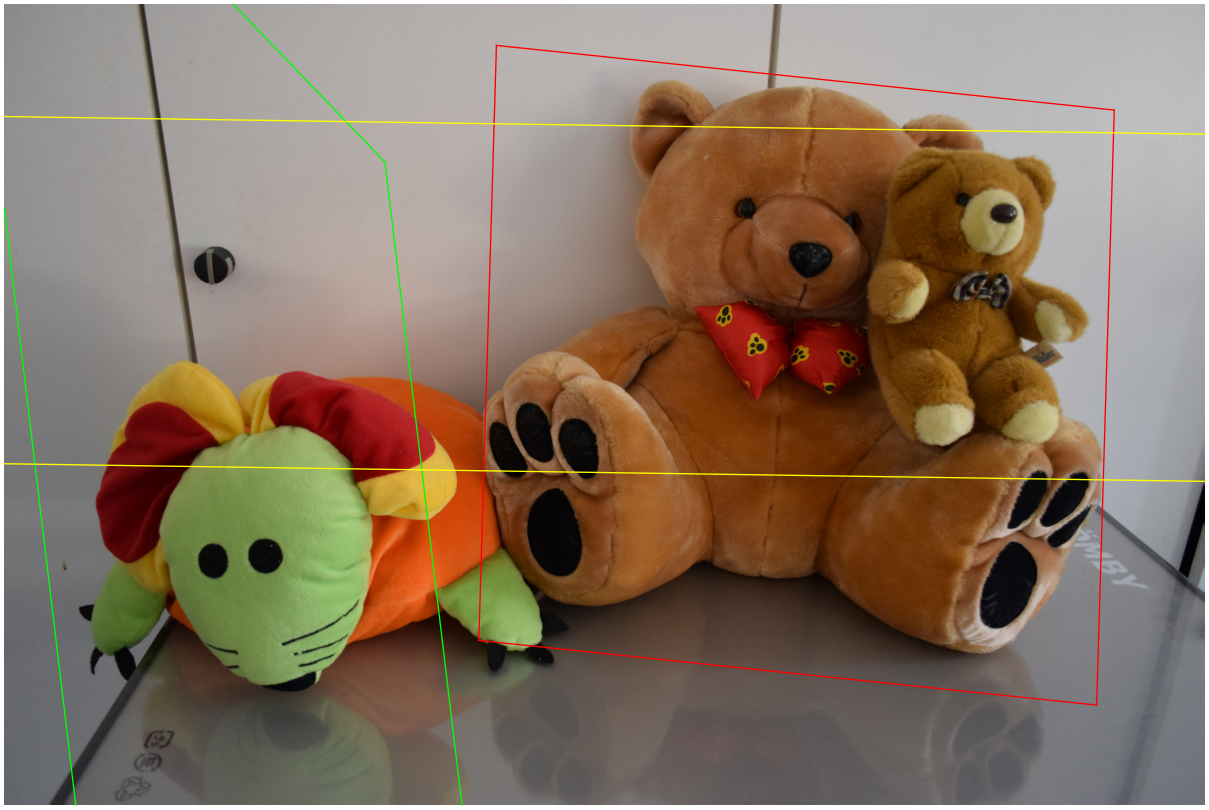

Figure 33: SURF method, dataset 2, scene 1

Figure 34: SURF method, dataset 2, scene 2

Figure 35: SURF method, dataset 2, scene 3

Figure 36: SURF method, dataset 2, scene 4

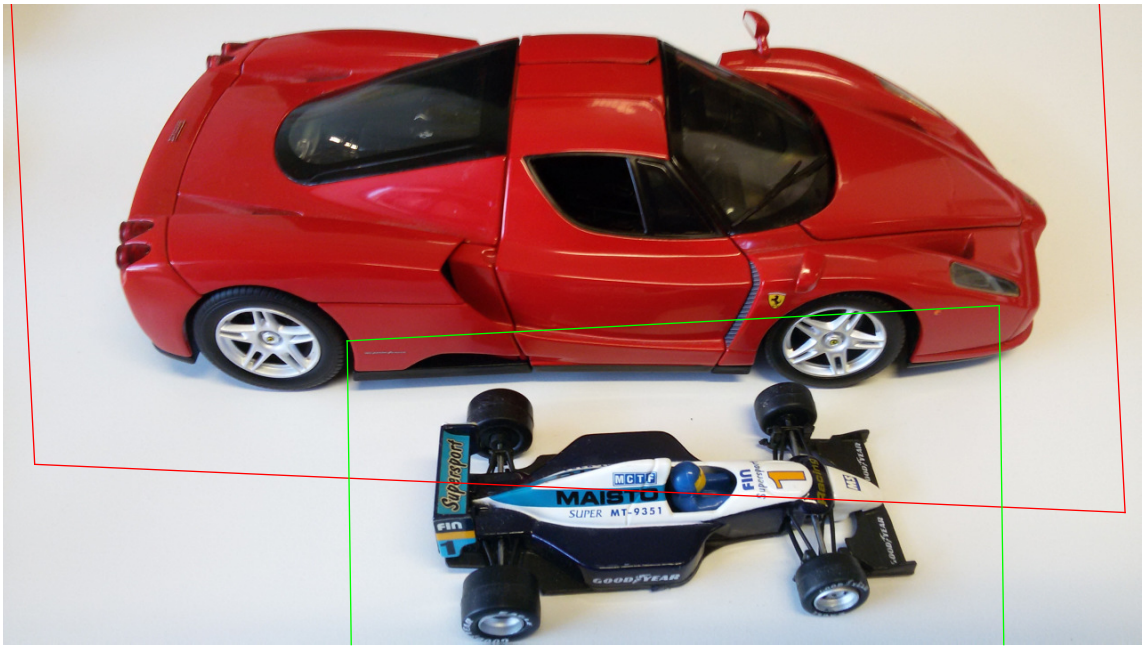

Figure 37: SURF method, dataset 3, scene 1

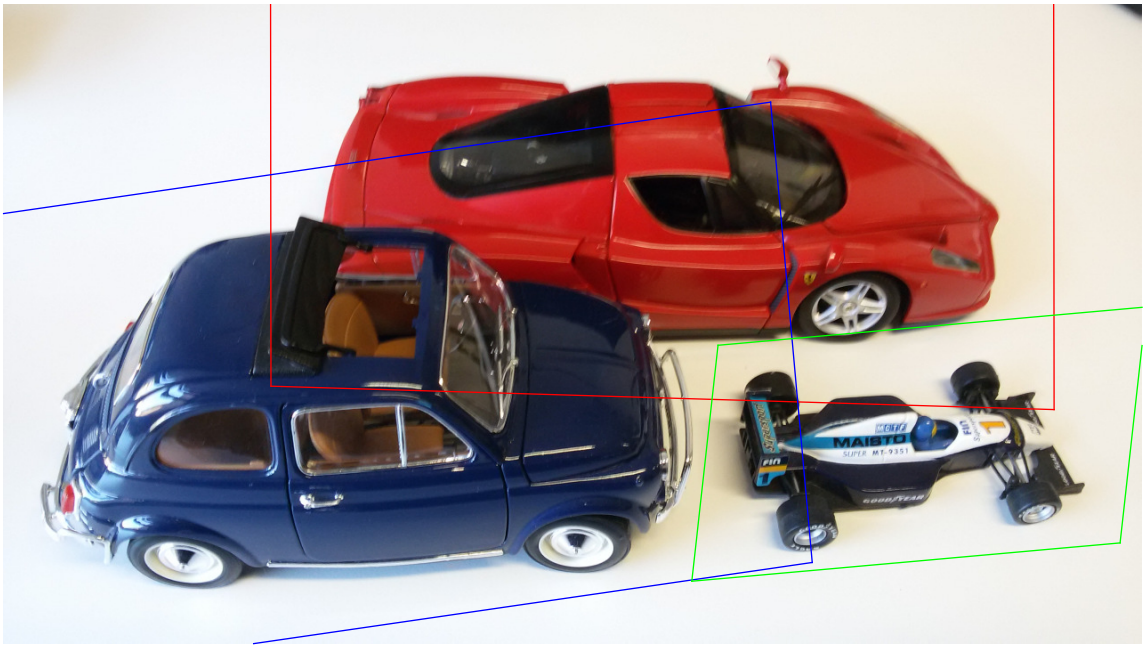

Figure 38: SURF method, dataset 3, scene 2

3.3.2 LBP

LBP algorithm is mostly used for texture recognition (as originally proposed in [9]) or face detection (as in [10] or [11] for example); one of its natural application is to recognize an object among a set of other images of objects.

The baseline approach seen in class is not rotational invariant, hence some extensions were added. The method implemented in Maltab `extractLBPfeatures` allows to select the number of neighbors (chosen in a circularly symmetric pattern) used to compute the LBP around each pixel, the radius of the circular pattern, the possibility to have or not rotation invariance and the size of the cells in which we want to divide the image considered. Note unfortunately that it is not possible to compute the Local Binary Patterns pixel by pixel because the Matlab's function does not allow it.

In Figure 39 we can see an example of application of the LBG algorithm to detect an object among others. Each bin represents a squared difference between the LBP description histograms of each object: a low value means that the similarity between the object is very high (in this case one comparison is always zero because it is actually the same image unless geometric rotation).

Furthermore we can think of exploiting this extended method in order to detect and localize an object inside a scene; however we should expect quite lower results. The main idea is

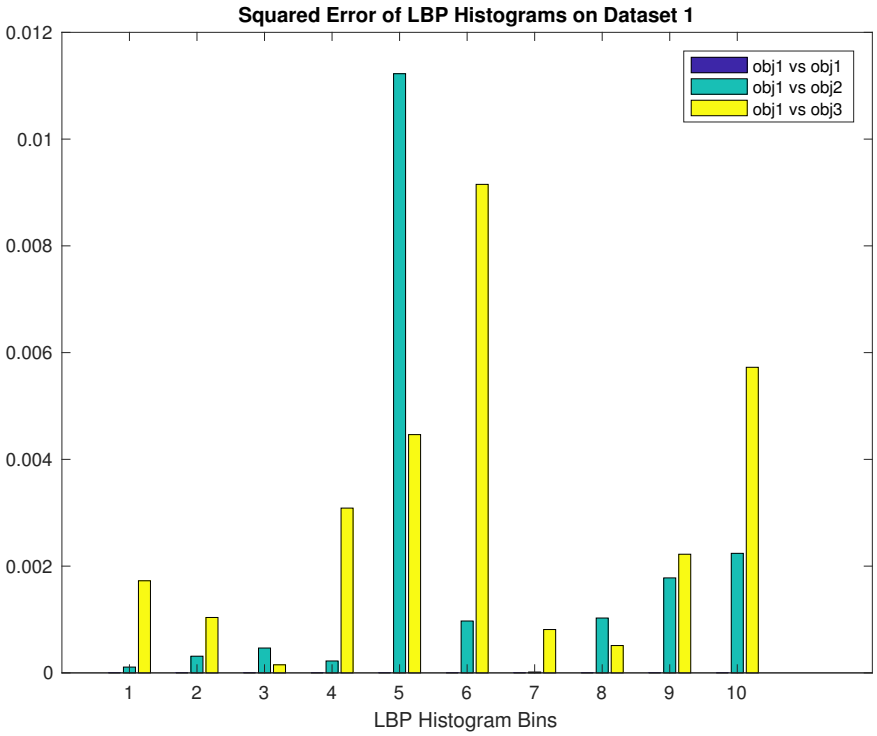

Figure 39: Example of LBP method for object identification with the objects of the first dataset, where some of them can be rotated.

to subdivide the images of the scene and of the object in non-overlapping cells each of a chosen dimension. Then extract the LBP features from each cell (i. e. a histogram with $\text{num. of neighbors} + 2$ bins for every cell in the image) and try to match the features of the scene with the ones of the object using e. g. a XOR operation or a thresholding distance criteria. From this point onwards we can re-apply the RANSAC algorithm in order to determine the affine transformation with the largest compatible set and select it as the actual one.

I implemented this procedure trying different configurations of the parameters (matching threshold, distance criteria, cell size, number of neighbors, number of iterations of the RANSAC and so on) but none of them leaded to acceptable results. However, since we were expecting results lower than the ones obtained with the SIFT or the SURF, I decided to postpone the improvements of this algorithm for future works.

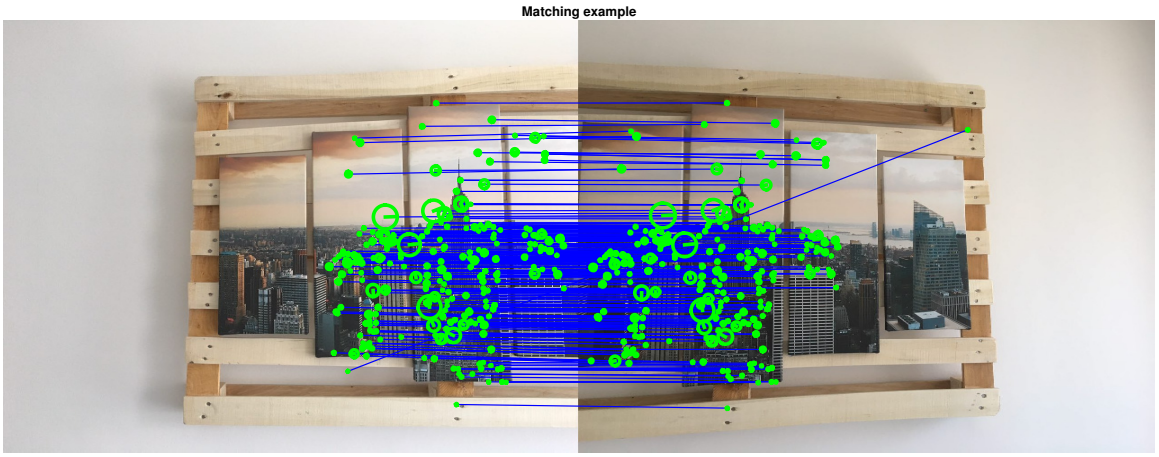

Figure 40: SIFT matches between two images of almost the same scene (*my dataset 4*).

4 Mosaicing with SIFT

Moreover, I decided to implement a very basic version of two images stitching using the SIFT features. The idea is to exploit the information of the positions of the SIFT matching features in order to estimate the mean shift in the horizontal and vertical directions. Note that here I assumed not to know the field of view of my camera, thus I decided not to project the images on cylindrical surface (differently from the fifth laboratory experience).

Here the main script is `mainMerging`, which provides an example of the procedure on a new dataset (*my dataset 4*); it calls `getTranslationSIFT(img1,img2,thresh,T)` which computes the most probable shift along the x and y directions (using the translation which leads to the largest compatible set) given the two images, the matching threshold *thresh* (which can be set around 0.2 % 0.4 because we have a large number of correct matches since the two images shows almost the same scene) and the threshold of the RANSAC algorithm T (which can be set very small for the same reason above).

The two images considered and the respective SIFT matching features are reported in Figure 40. To confirm what we said above, we can notice the high number of correct matches and just a few wrong ones.

Applying the idea already discussed, we obtain pretty good results considering also the high generality of this method since we do not even specify the field of view of the camera (see Figure 41). Of course there are some artifacts at the juncture of the two images and the overall sensation appears to be distorted because the perspective between the two images is not the

Figure 41: Merged images of my dataset 4.

same (the object in the scene was maybe too close to the camera, about 1 meter far). If we want better results we need to specify the field of view of the camera and to apply the cylindrical projection of the images to be merged (as already done in laboratory 5).

5 Brief Analysis on Videos

In this section I would like to extend some tasks of the motion estimation of videos.

First of all I have implemented a script in order to find if there is motion or not in the video by looking at the largest connected set of detected moving objects (using the code for the seventh laboratory experience and the fixed threshold method with $T = 0.1$, which is enough for our purposes). The main script is `video7.m` which calls the function returning if there is movement in the current image or not: `isMovement(BWimage, threshold)` where *BWimage* is the black and white image obtained in output from the fixed threshold algorithm and *threshold* is the minimum dimension for a connected set to be considered as a moving object. This last parameter allows us to adapt the code to very different situations achieving the desired trade-off between accuracy (size of objects we want to detect but also expected speed of the objects) and robustness.

I applied this code to the first video taken in the laboratory (`lab2017-1.mp4`) and the result was very precise and reliable; a screenshot can be seen in Figure 42 with a threshold of 200 connected pixels but the code can be easily ran to see frame by frame if there is movement or not (displayed to command window while the image is updated every 0.5 seconds in order to realize what is happening).

Moreover, since a very important application is intrusion detection in surveillance cameras, I applied the code to the `surveillance.mpg` video and a screenshot is reported in Figure 43 using a threshold of 50 connected pixels since the moving objects are smaller and slower than before.

Many improvements can be done and the code is already very lightweight and can be safely evaluated in real-time. For example we can enhance the performances by using more sophisticated methods for movement detection (such as running mean or probabilistic approaches) and we can think of adaptive threshold mechanisms and so on, but the results are already quite satisfactory.

Finally I implemented a basic algorithm in order to estimate the motion using also the past history of the corners extracted using the Harris algorithm. To this purpose I wrote the script `video8.m` which is structured similarly to the code of the eighth laboratory experience. Here you can choose the desired frame on which draw the history of the corners of the previous

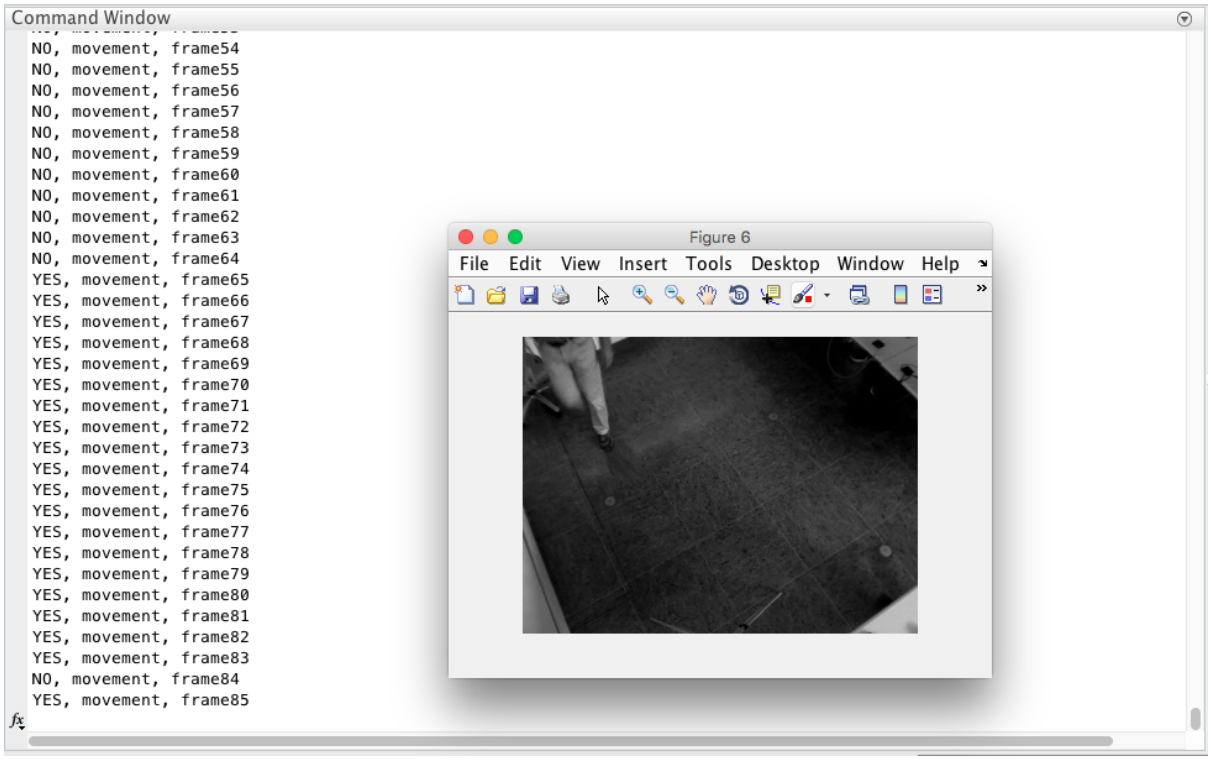

Figure 42: Detection whether moving objects are present into the scene, `lab2017-1.mp4`.

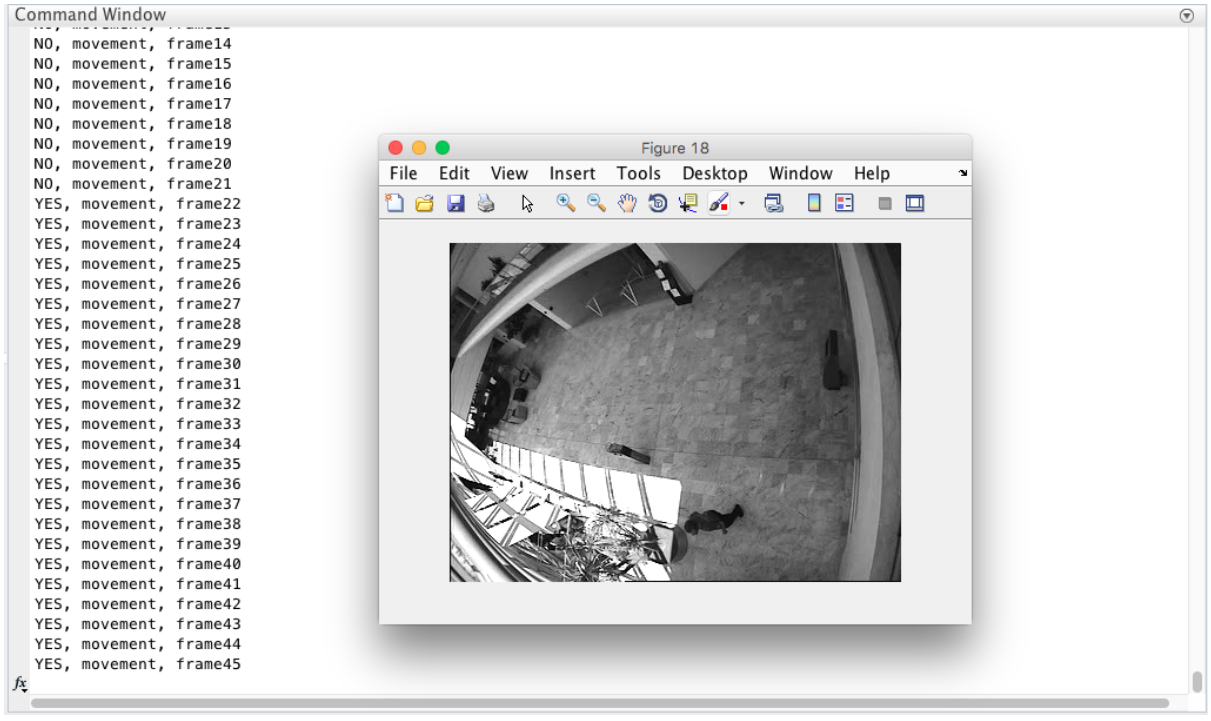

Figure 43: Detection whether moving objects are present into the scene, `surveillance.mpg`.

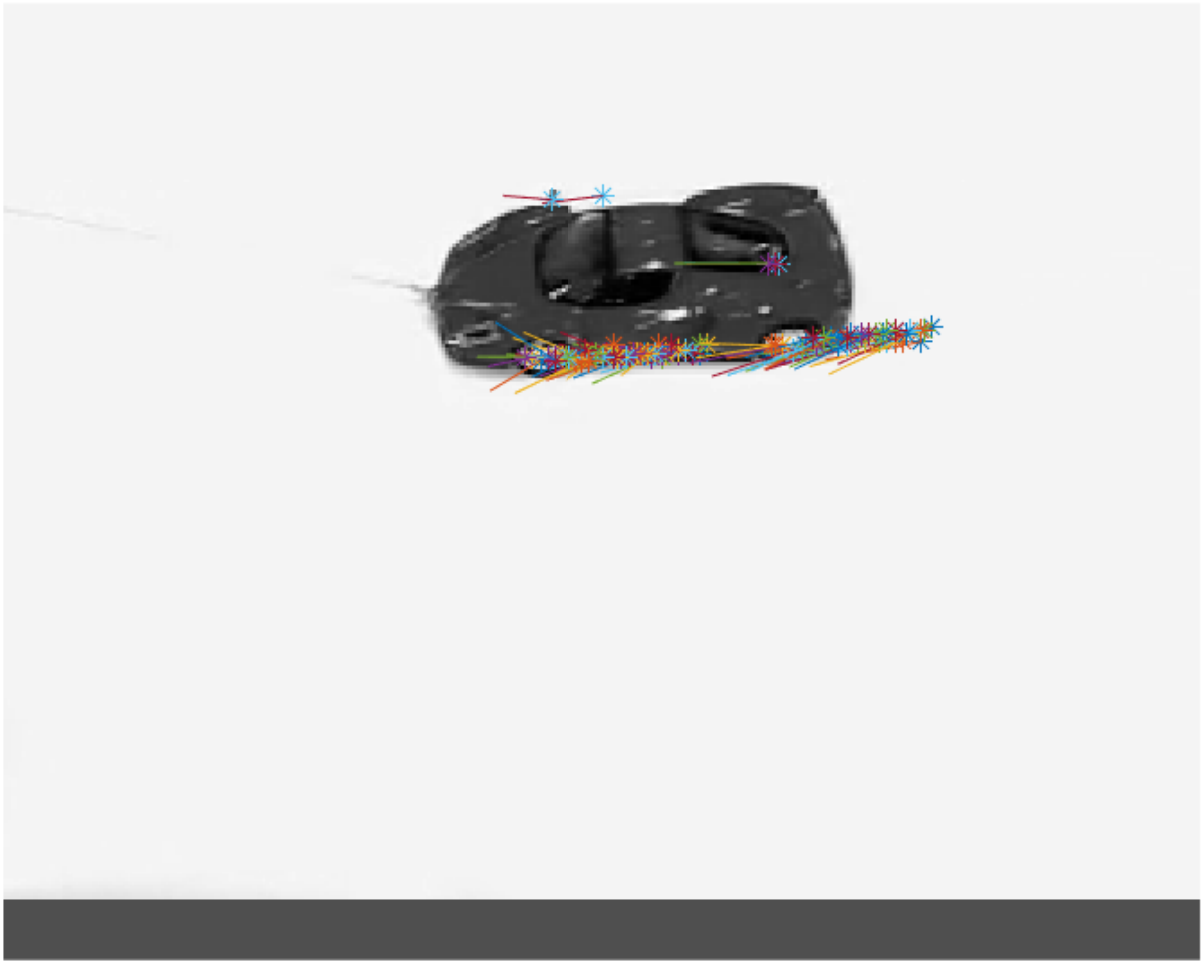

Figure 44: Frame number 100 with 20 frames of history of the 4 strongest corners.

frames (you can also specify how many corners and how many previous frames do you want in order not to make the computations very heavy).

In Figure 44 we can see some frames of the `car1.mp4` video with just the four strongest corners for each frame; the result is very reliable and tell us the path traced by the car and the directions of the corners. In this simple example the car was tracing a straight path from right to left.

In Figure 45 a more challenging video is examined `crossing_cars.mp4` using the ten strongest corners to visualize better the results. Here we can see the two paths of the cars and we can notice that the cars have just collided because the directions of the lighter car have been influenced with horizontal deviations.

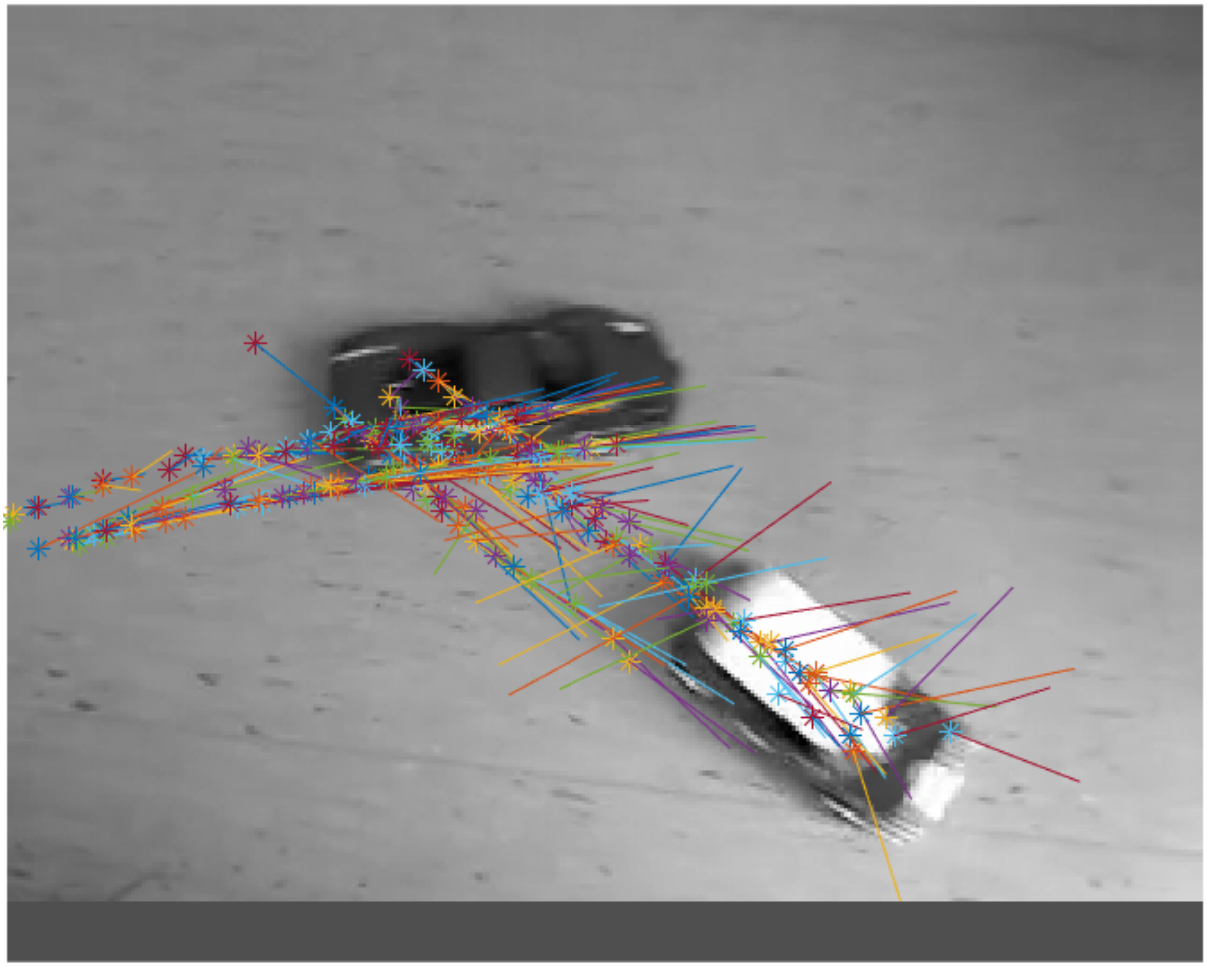

Figure 45: Frame number 130 with 20 frames of history of the 10 strongest corners.

6 Conclusions and Future Works

I found these Computer Vision's topics really amazing and exciting although it was just a taste of the actual possibilities of these algorithms. Many improvements can be further done in order to enhance the robustness and generally to get better results. Furthermore it still remains to correctly implement a later version of the LBP algorithm for object recognition.

Other works can be done in enlarging the feature descriptors algorithms involved in the comparison and compute some metrics such as execution time, quality of the results and dependency from the input parameters to fully compare them.

Also regarding the video estimation, one can think of applying a dynamic thresholding to detect the moving objects or more precise algorithms for moving objects detection; however it heavily depends on the targeted application desired.

My personal interests range from robotics to bio-engineering to autonomous vehicles and many others; I think that some arguments can be useful for my future career because can be applied in many different ways.

References

[1] D. M. Gavrila, "The visual analysis of human movement: A survey," *Computer vision and image understanding*, vol. 73, no. 1, pp. 82–98, 1999.

[2] G. R. Bradski, "Computer vision face tracking for use in a perceptual user interface," 1998.

[3] C. R. Wren, A. Azarbayejani, T. Darrell, and A. P. Pentland, "Pfinder: Real-time tracking of the human body," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 19, no. 7, pp. 780–785, 1997.

[4] A. J. Lipton, H. Fujiyoshi, and R. S. Patil, "Moving target classification and tracking from real-time video," in *Applications of Computer Vision, 1998. WACV'98. Proceedings., Fourth IEEE Workshop on*, pp. 8–14, IEEE, 1998.

[5] M. Brown and D. G. Lowe, "Automatic panoramic image stitching using invariant features," *International journal of computer vision*, vol. 74, no. 1, pp. 59–73, 2007.

[6] S. Se, D. Lowe, and J. Little, "Vision-based mobile robot localization and mapping using scale-invariant features," in *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, vol. 2, pp. 2051–2058, IEEE, 2001.

[7] D. G. Lowe, "Object recognition from local scale-invariant features," in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, vol. 2, pp. 1150–1157, Ieee, 1999.

[8] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Computer vision and image understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[9] T. Ojala, M. Pietikainen, and T. Maenpaa, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 24, no. 7, pp. 971–987, 2002.

[10] T. Ahonen, A. Hadid, and M. Pietikainen, "Face description with local binary patterns: Application to face recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 12, pp. 2037–2041, 2006.

[11] X. Wang, T. X. Han, and S. Yan, “An HOG-LBP human detector with partial occlusion handling,” in *Computer Vision, 2009 IEEE 12th International Conference on*, pp. 32–39, IEEE, 2009.