Computer Vision 2021: Final project

Project 2

Topics: Object detection
Goal: Find the instances of an object in a scene using keypoint descriptors

Student: Grimaldi Alberto
Matricola: 2026704

Professor: Zanuttigh Pietro

Object Detection

Object detection in computer vision has the aim of detecting instances of object in digital images and videos. This kind of technology is mostly used for face detection and pedestrian detection. The aim of this project will be to implement software to detect correspondences between the features of the object and the features of the scene. We will later use these matches to locate the object in the scene.

Algorithm

The implemented algorithm is based on the following pipeline:

1. We read all the images from the selected dataset. To do this we use the cv::glob function which allow to read files in sequence from a directory. The images are converted in greyscale.
2. Next, we have to extract the keypoints and descriptors from the object image and from the scene. Both ORB and SIFT are implemented in the code.
3. Find the matching features.
4. Draw the bounding box of the object instance in the scene image.
5. Visualize the matches found and the object located in the scene.

Implementation

The Visual studio 2019 IDE and the C ++ programming language were used for the project. It is composed of 2 files: *"finalProject.cpp" and "obj_detc.h".*

The main library used in this project is OpenCV which is mainly aimed at real-time computer vision.

In the main() function we first select the dataset we want to use. Then we read all the images from the selected dataset by calling the *cv::glob()* function and running two *"for"* loop*:* one to read the

object images and the other one to read the scenes, putting them into separated arrays *"img_object" and "img_scene".* At the end of the main() we employ a nested "*for"* loop to compare each "object" image with each "scene". For this end one of the two functions can be applied: ***objDetec_SIFT(image_object, image_scene, threshold)*** *or* ***objDetec_ORB(image_object, image_scene, threshold),*** based on the desired feature extraction method.

In objDetect_SIFT() function we first extract the keypoints using SIFT detector and then compute the descriptor using *detectAndCompute* method of the *cv::Feature2D* class. In the next step we match the descriptors through *Brute-force matcher*. It takes the descriptors of one feature in first set and is matched with all other features in the second set using the L2 distance (Euclidian distance). And the closest one is returned. We set *true* the second parameter of *BFmatcher* to provide a more confident result. To refine the matches found (good matches) we apply a ratio test by selecting the matches with distance less than *threshold* * *min_distance*, where threshold is the third parameter of objDetect_sift() function, and min_distance is the minimum distance found among the matches. *Cv::drawMatches()* helps us to draw the matches. It stacks two images horizontally and draw lines from first image to second image showing best matches. We localize the object in the scene by applying the function ***cv::findHomography()*** with method=RANSAC, which returns 3x3 matrix with the computed homography. Then we extract the corners of the object image. We need corners to draw the lines on the mapped object in the scene. Through the ***perspectiveTransform()*** function we apply the homography H on *obj_corners* vector and put the results in *scene_corners* vector. Finally, we draw lines on the mapped object in the scene.

Functions objDetect_SIFT() and objDetect_ORB() are very similar. The first one implements the SIFT detector and the second one the ORB detector. Another difference is related to the matching algorithm: since ORB is a descriptor based on binary strings we used the Hamming distance instead Euclidean distance.

Results

Now some results will be shown and a comparison between ORB and SIFT will be made.

Let us start by showing the threshold effect. We note that by increasing the threshold value from 3 to 6, the number of matches found is greater.

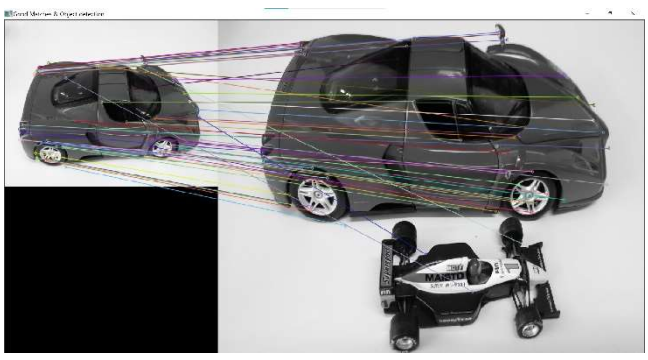


Figure 1: SIFT, threshold 3

Figure 2: SIFT, threshold 6

Here we can observe how the SIFT detector and the ORB detector work differently. Although we are using the most difficult dataset, SIFT detector can generate very accurate matches. ORB instead generates many wrong matches.

Figure 3: SIFT, threshold 3

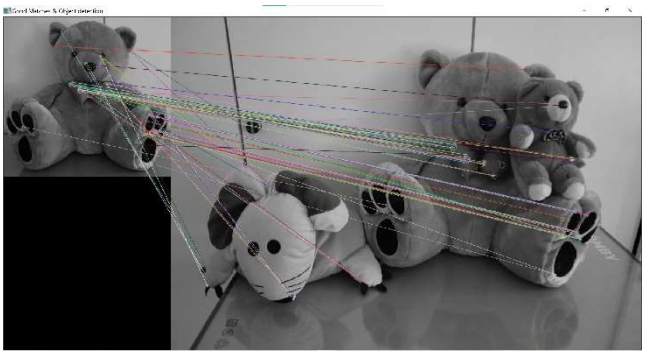


Figure 4: ORB, threshold 3

Running the algorithm until dataset3, the SIFT detector does not provide any wrong match. With ORB , instead, the feature matching task become more challenging.

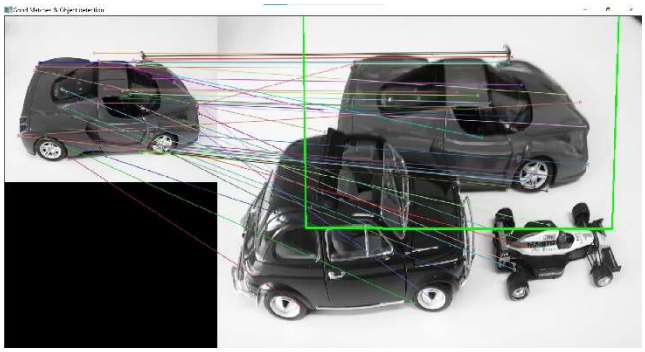


Figure 5: SIFT, threshold3

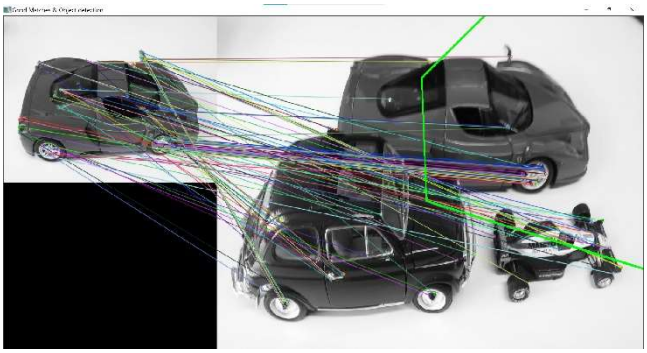


Figure 6: ORB, threshold 3

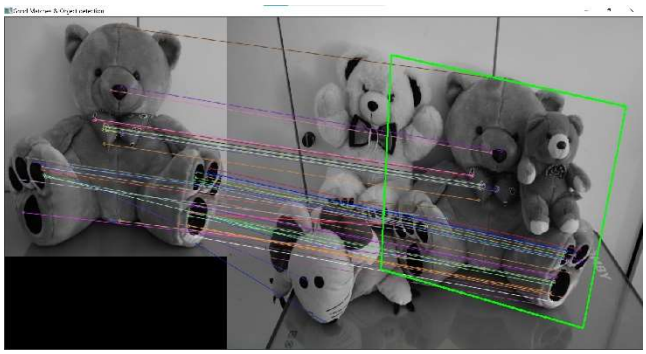


Figure 4: SIFT, threshold3

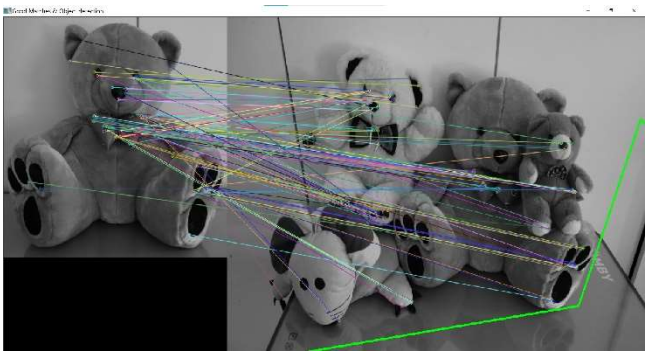
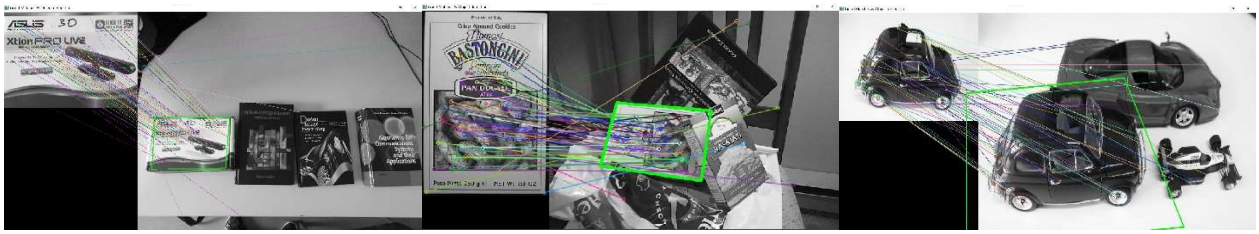
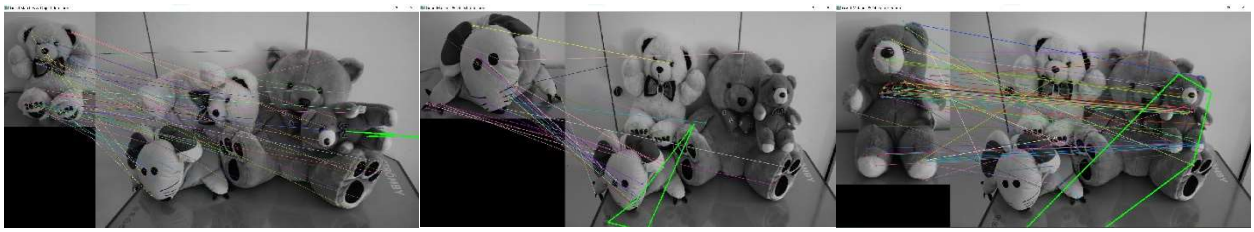


Figure 3:ORB, threshold 3

Finally, below we can see some good results and some wrong matches detected by SIFT detector. Most of the wrong ones belong to the last dataset, the most difficult one.





To complete the report, a brief performance analysis will now follow. The results obtained using the SIFT detector show a rate of 100% of matches correctly detected for datasets 1,2 and 3, while a rate of 60% of matches correctly detected (8 out of 13) for the most difficult dataset. With the ORB detector the performances are lower: we have a rate of 100% correct matches for the first dataset, 33% (1 out of 3) for the second dataset, only one wrong match for the third dataset and finally a rate of 38% correct matches (5 out of 13) for the last dataset.