REPORT SECOND ASSIGNMENT - COMPUTER VISION

The goal of this project is to implement and test the performances of at least two different feature detectors (in this case, Good Features to Track, SIFT and ORB) complemented with a tracking algorithm (in this case, Lucas-Kanade optical flow algorithm).

- main.py: this file is structured in an iterative way such that at each iteration the same video Contesto_industriale1.mp4 is captured and is passed as a parameter to the 3 implemented algorithms. Moreover, a check is carried out on the correct functioning of the video and on the reading of the first frame. Let's have a look to the 3 algorithms.

- GFT_and_OpticalFlow.py: the first detection algorithm that works on the aforementioned video is the Good Features to Track: it is a function that finds the most prominent corners in an image or in a specified image region, thus determining strong points of the current frame. The function can be used to initialize a point-based tracker of an object; indeed, the detected points are passed to the tracking algorithm Lucas-Kanade optical flow which calculates an optical flow for a sparse feature set using the iterative Lucas-Kanade method with pyramids.
  As for the implementation structure, the first frame of the video is initially opened and transformed into grayscale values, as well as declaring the parameters that will be passed to the two algorithms. After detecting the keypoints (*corners*) in the first frame using the *goodFeaturesToTrack* function, we enter the for loop which will end when a maximum number of frames (*MAX_FRAMES*) is reached, or if no more available frames are found, or if we press *'q'* on the keyboard.
  At each iteration the following operations are performed:
    - a new frame is extracted and immediately converted to grayscale values;
    - the keypoints corresponding to those of the previous frame are tracked on the current frame using the *calcOpticalFlowPyrLK* function (the parameter *corners* represents the vector of the points for which the flow needs to be found);
    - at this point, if the number of corresponding tracked points is less than the fixed minimum number (*TRACKED_ POINTS*) of remaining tracked keypoints along the frames, new corners are detected with the Good Features to Track algorithm which are passed to the Lucas-Kanade optical flow algorithm to restart the tracking starting with them; otherwise, the corners of the previous frame continue to be tracked;
    - then, we draw the tracking lines and the detection points on the current frame, using the function *drawKeypoints* (described below), which is shown in output.
    - finally, the frame and corners are updated for using them in the next iterative step.

  Once out of the for loop, the video is released, and everything is destroyed.

- SIFT_and_OpticalFlow.py: the second detection algorithm implemented is SIFT, a method for extracting keypoints and computing descriptors using Scale Invariant Feature Transform, which is complemented with the Lucas-Kanade optical flow tracking algorithm. In this case, the extraction of keypoints in the image is used for both tracking and matching in the image. The implementation structure is quite similar to that of *GFT_and_OpticalFlow.py*, but there are various differences between the two detection algorithms. Let's go see them.
  Again, the first frame is opened, converted to grayscale values, and keypoints are detected and associated descriptors are computed on it using SIFT's *detectionAndCompute* function. Here we have a substantial difference with the previous algorithm: it is necessary to extract the coordinates from the keypoints in order to subsequently track them using the *keypoint_into_coordinates* function (described below). At this point we enter the previously described for loop and, at each iteration, we work on a new frame converted into grayscale values. On the current frame:
    - new keypoints with associated descriptors are detected, and
    - the points corresponding to those left in the previous frame are tracked.

  If the latter number of tracked points is less than the fixed value *TRACKED_POINTS*, the new keypoints detected are transformed into pairs of coordinates and passed to the *calcOpticalFlowPyrLK* function which restarts the tracking on them; otherwise, it continues to track the keypoints from the previous frame. Again,

the tracking lines and points detected are drawn (*drawKeypoints* function) on the current frame. Another substantial difference compared to the Good Features to Track algorithm is that with the SIFT algorithm we can find any matches between the detected features along two consecutive frames. The best (based on Euclidean distance) matches are extracted and drawn for each pair of previous and current frame and then the output is launched showing both the detection algorithm matches and the tracking algorithm lines. As before, the parameters for the new iterative step are updated, and once the for loop is finished, the video is released, and everything is destroyed.

- ORB_and_OpticalFlow.py: the latest detection algorithm is ORB (Oriented FAST and rotated BRIEF), an efficient alternative to SIFT or SURF algorithms used for feature extraction, in terms of computation costs, matching performance and mainly patents. The algorithm uses FAST in the pyramids to detect stable keypoints, selects the strongest features using the FAST or Harris response, finds their orientation using first order moments, and computes descriptors using BRIEF (where the coordinates of pairs of random points are rotated according to the measured orientation).
  The implementation structure is identical to that of *SIFT_and_OpticalFlow.py*, with the difference that the *ORB_create* function is called instead of *SIFT_create* and in the *BFMatcher* object the normType parameter is *NORM_HAMMING*, which used the Hamming distance as a measure, instead of *NORM_L1* (good for SIFT).

- Utility.py: this file contains two useful functions for the 3 algorithms previously described:
  - *keypoint_into_coordinates*: it is useful to obtain the coordinates of the keypoints detected by the SIFT and ORB algorithms to pass them to the tracking algorithm.
  - *drawKeypoints*: it draws a line connecting each pair of tracked points (the old keypoint and the corresponding new one), and also draws the circle of the new point detected in the current frame; this function returns the final frame with tracking.

Three outputs are shown in sequence in the following order:

- a sequence of current single frames containing the tracking lines of corners detected with the Good Features to Track algorithm;
- a sequence of pairs of consecutive frames such as to highlight the matches of the keypoints detected by the SIFT algorithm, in addition to the tracking line in the current frame;
- the same output above, but with the ORB detection algorithm, instead of the SIFT one.

In all three cases, the tracking algorithm is Lucas-Kanade optical flow.