Feature Detection and Tracking Assignment Report

Simone Alghisi (229355), *Master Student of Artificial Intelligence Systems*

I. Introduction

THE goal of this assignment was to implement and test some feature-based detection and tracking algorithms. In particular, the idea was to experiment as much as possible on a given video with different approaches, fine-tuning the parameters to enhance the (qualitative) performance in a particular real-case scenario.

A. Analysis of the Video

The footage proposed for the testing is very peculiar because:

- objects have very similar characteristics (features);
- both the background and the foreground are particularly flat;
- all the objects in the scene behave similarly (i.e. are influenced only by the camera movement apart from small possible lens distortions);
- some components are characterised by glossy surfaces, introducing problems due to possible reflections;
- illumination is overall constant, aside from small LEDs.

Fig. 1. One of the first frame of the video.

II. Proposed Methods

From the preliminary analysis explained in Section I-A, it can be quickly understood why region-based techniques would easily fail (or at least would make both detection and tracking much more difficult). Indeed, the suggested approach to use feature-based techniques makes more sense.

To this end, the following methods were considered for feature detection:

- Good Features to Track (GFTT), which highlights those points that exhibits high gradient values on both the X and the Y direction. In particular, it considers textures and corners;
- Scale Invariant Feature Transform (SIFT), which - as the name suggests - finds those keypoints that are scale (and rotation) invariant (i.e. unaffected by scale) and the descriptors associated with them;
- ORB (Oriented FAST and Rotated BRIEF), that is a fusion of FAST keypoint detector and BRIEF descriptor with many modifications to enhance the performance, making it faster than SIFT while maintaining (almost) the same accuracy.

Starting from these detectors, the following algorithms were considered for tracking/matching:

1) Lucas-Kanade Optical Flow;
2) Brute-Force Matcher;
3) Kalman Filter.

Finally, in order to investigates other methodologies a template-based approach was considered.

III. Results

In this section, a qualitative analysis of the results is presented for each approach that was considered, along with the possible variations that were put in place to reach better performance.

A. Lucas-Kanade Optical Flow

Lucas-Kanade Optical Flow was the very first technique to be considered, mostly because we already had the opportunity to experiment with it during the lab sessions. In order to compute the matching, the algorithm requires both the frames at time $t-1$ and t, and the features detected at the previous iteration. Due to this, it is necessary to initialise the procedure using a feature detector; in particular, I have decided to run some experiments using GFTT, SIFT, and ORB.

Stating the difference between the three feature detectors is actually quite difficult, because in all three cases the matching was performed with almost flawless precision. However, it is possible to notice how SIFT manages to be more consistent with the features extracted across different objects, due to the fact that is unaffected by rotations and scaling.

Unfortunately, SIFT comes at a great cost in terms of computational requirements, and causes the video to slow down. A good trade-off can be reached by decreasing the number of features that are extracted by the detector. Of course, other parameters can be controlled to increase the overall performance, such as the number of layers for each octave, or the sigma value which is used for the Gaussian smoothing. At the end, when working with 30 features and an

TABLE I
SIFT & ORB COMPARISON

Detector	N. Features	Time (MM:SS)	CPU (%)
SIFT	100	01:11	361
ORB	100	00:41	343

initial sigma level of 3, SIFT is still able to reach really good accuracy.

Instead, another approach consists of using ORB, which is faster and can flawlessly deal with thousands of features. However, speed comes at a slight less quality and, for this reason, it is possible to observe that some keypoints detected are not as good as the one found by SIFT: in fact, they are sometimes associated to borders and tend for the majority to overlap with each other.

Finally, I considered the GFTT algorithm an noticed very interesting results when tweaking the parameters correctly. In fact, with the default configuration that we have used for the lab some flat surfaces/lines are detected, leading to oscillations during the tracking. However, despite being a simpler algorithm (i.e. a corner detector), by specifying an higher quality and an higher minimum Euclidean distance it is possible to detect very stable and uniformly distributed keypoints.

At the end, a final remark is related to the sampling rate. Indeed, Lucas-Kanade purpose is to track features across two consecutive frames (i.e. the optical flow assumption should hold). However, features can be occluded as time goes on; in particular, the camera focus in the proposed video shifts from right to left and so features start disappearing. To solve this issue, features need to be periodically detected and this means that some algorithms, such as SIFT, could become computationally very heavy. In this case, it is possible to have good performances by sampling every 50 frames (minimum 70).

B. Brute-Force Matcher

Given that descriptors associated to keypoints are required for the Matcher, I have decided to consider both SIFT and ORB detectors for such a task. In particular, I wanted to evaluate how much difference there is between the two in terms of computational power required. In fact, in this particular scenario it is necessary to retrieve the descriptors at every frame in order for the matching to be effective, which has an impact on the real-time performance. Moreover, to compare the Brute-Force Matcher performance w.r.t. Lucas-Kanade, I have decided to keep one of the set of descriptors fixed, updating it only every 50 frames.

As expected, SIFT has a very strong impact on real-time performance. In fact, as shown in Table I, the time required for processing the video increases by 30 seconds. Of course, as anticipated in Section III-A, the results obtained with SIFT are qualitatively better compared with ORB.

Overall, Brute-Force Matcher is very computationally demanding because descriptors need to be computed at every frame and the matching procedure is indeed a brute force approach. Moreover, results obtained are worse compared to Lucas-Kanade. In fact, previous keypoints are never discarded, and this means that the same descriptor could match a similar keypoint in a completely different location as shown in Figure 2

Fig. 2. From Left to Right: the keypoints extracted previously; the matching keypoints in the current frame.

C. Kalman Filter

The third proposed technique is the Kalman Filter, i.e. a probabilistic approach to estimate the next position of a given entity - which is assumed to have certain properties at a given instant (i.e. location, speed, acceleration) - subject to some noise.

Of course, in this scenario multiple objects are present, which means that different matrices (i.e. measurement, transition, process noise, and measurement noise) should be used to track each of them. However, the objects in the scene behave all in the same way: in particular, they are all subject to the motion of the camera. For this reason, a single set of matrices can be used to estimate the motion of all the keypoints in the current system.

Unfortunately, a strong limitation of the system is the fact that the only observations that can be used in order to update the starting matrices are the ones provided by another tracking algorithm, namely from Lucas-Kanade. This means that, at the end, the system is already tracking the object that Kalman should track.

Nonetheless, the results are quite interesting and it is possible to see how (provided enough observation) the process stabilizes after some iterations, providing predictions that are very close to the observations as shown in Figure 3.

In order to reach such performance, several trials were performed to properly tweak the starting matrices. Moreover, when few observations (features) are available, Kalman is not able to perceive the scene as a whole and correctly predict the next point position. Such behaviour is probably due to the fact that Lucas-Kanade observations are very consistent and, after some frames, the keypoints are updated, meaning that they could jump abruptly from one position to another in a non-linear fashion.

Fig. 3. From Left to Right: Kalman at the first frame; Kalman after some frames

Fig. 4. Multiple template matching applying binarisation as pre-processing.

D. Multiple Template Matching

Lastly, I decided to tackle the problem from a different perspective; in particular, I used a template-based matching procedure in order to detect some specific objects in the scene. Of course, I knew from the very beginning that this method was not the best possible approach, due to the fact that it is very scale, shape, and also color dependent.

Indeed, the first results were not convincing at all. For this reason, I decided to improve its accuracy by employing some very simple pre-processing techniques; in particular, I first tried to use inverse binarisation in order to focus on darker objects. However, illumination - mostly shadows - proved to be a problem, which was addressed by applying a simple Gaussian Blurring. Finally, I also tried the Canny edge-detector in order to focus more on corners.

Unfortunately, such simple techniques did not lead to astonishing improvements: in fact, the algorithm is still quite weak to illumination and scaling. Nonetheless, I was surprised about how, when simply considering the starting template the procedure was able to correctly detect many objects.