Feature Detection and Tracking Assignment 2

Ambrosini Matteo
Artificial Intelligence Systems
Povo, Trento, Italy
matteo.ambrosini@studenti.unitn.it

I. What is Feature Detection?

In Computer Vision, a **feature** is a piece of information about the content of an image. Features are typically explored to assert whether a region of an image has certain properties. More broadly a feature is any piece of information which is relevant for solving the computational task related to a certain application.

Feature detection is a **low-level** image processing operation. That is, it is usually performed as the first operation on an image, and inspects pixels to see whether there is a feature present in that region or not. Because feature detection algorithms search into each and every pixel, they are usually employed as part of a larger algorithm. This way the algorithm will typically only examine the image in the region of the features. As a built-in prerequisite to feature detection, the input image is commonly smoothed by a Gaussian kernel in a scale-space representation and one or several feature images are computed, often expressed in terms of local image derivative operations and computed by means of operators such as the Laplacian, Sobel, Roberts and Prewitt.

Once features have been detected, they are (almost) ready to be extracted. Note that such an extraction may involve considerable amounts of resources, and for this reason there are techniques that are engineered specifically to reduce computational burden whilst maintaining a high level of accuracy. **Intensity** and **direction** of edges is one of the most salient features that can help us to characterize objects.

II. Workflow

Feature extractors used in this project are:

- **AKAZE**: a local binary descriptor showing improved results in terms of speed and performance compared to state-of-the-art methods such as local feature descriptors.
- **ORB**: it uses FAST to find keypoints, then applies Harris corner measure to find the top N keypoints. It is rotation invariant.
- **SIFT**: applies an increasingly intense Gaussian blur filter to the 'train' image. It performs pixel-wise difference between octaves to look for the most intense variation in value. SIFT is both rotation and scale invariant.

Trackers used in this project are:

- **BFM**:
- **FLANN**:

Each and every tracker is tested with each and every feature extractor. Such an experiment is hopefully able to highlight any variation in performance due to unique matches. **Performance** is evaluated by keeping track of both the **framerate** and the number of **good features** detected.
Good features are obtained following the guidelines of Lowe's paper. In a nutshell, only the two best matches for each keypoint – hence the two keypoints with the *smallest distance* measurement – are considered to be significant. Lowe's test also checks that the two distances are *sufficiently different*. If they are not, then the keypoint is eliminated and will not be used for further calculations.

III. Performance Evaluation

Detectors

	Average fps	Feats per frame
AKAZE + BFM	1.41	5006.70
ORB + BFM	7.36	6754.30
SIFT + BFM	1.3	3686.72
AKAZE + FLANN	0.65	2008.15
ORB + FLANN	7.58	8674.25
SIFT + FLANN	1.38	1000.17

Whenever paired with BFM, detectors like SIFT and AKAZE are able to find a number of features in line with ORB. However, in an online application they may struggle a lot with keeping up with the frame rate. For this reason we tried to set a *cap* to the number of features to be detected, however both algorithms are still stuck at just above 1 fps.
As mentioned in the Workflow section, we carried out our second experiment pairing each and every feature detector with FLANN matcher. The result yields values that are fairly consistent with BFM. SIFT and ORB are along the lines of their match with brutal force matcher as far as framerate is concerned. However, ORB shows a surge in the number of features it is able to detect. This is almost certainly due to the different approach in storing and displaying good matches. (see the code for reference).

IV. Conclusions

In conclusion, the performances of AKAZE, SIFT, and ORB are almost equivalent in terms of speed whenever it is the case that we are dealing with static material like images (see standalone files in the code). However, if we need to deal

with an online scenario like the one in analysis, the gap in performance between ORB and any other detector analyzed becomes unbridgeable.
One more thing which needs to be taken into consideration is the application field of each detector. For instance, comparing the 'train' image of an object with a 'query' image of another object of the same category differs a lot to comparing a 'train' image of that object with a 'query' image containing several other objects of the same category.
Bearing this concept in mind, we compared subsequent frames of the same video containing fairly the same number of objects.

V. Links

- Github
- Demo Video
- Presentation Video