Mid-Term Project Report

by

Kelechi Osuji

MP.1 Data Buffer Optimization

This was achieved using a ring buffer of size 2, which holds the two images whose keypoints' descriptors are to be matched in each iteration through the 10 images. In the implementation of the ring buffer, we first check if the size of the buffer has been reached or exceeded. If that is the case, we first delete the oldest element in the buffer, and after that, we load a new image into the buffer. But if it is not the case, we load the buffer with an image. With this, image1 and image2 are first loaded, matched, then image1 is deleted, image3 is loaded, image2 and image3 are matched, and so on. And this circle continues until the last image. However, the good thing is that the elements in the buffer remain maximum of 2 from beginning to the end of the whole process, which saves memory.

MP.2. Keypoint Detection

Using OpenCV I implemented the HARRIS, FAST, BRISK, ORB, AKAZE, and SIFT detectors and made them selectable with 'if else' statements using respective strings.

MP.3. Keypoint Removal

Given the box parameters which is based on the 'Rect' datatype in OpenCV, I used its contains() method to extract only the keypoints that are located within this rectangle in which the preceding vehicle is located, i.e. all other keypoints outside the box are ignored.

MP.4. Keypoint Descriptors

Using OpenCV, I implemented the BRIEF, ORB, FREAK, AKAZE and SIFT detectors and made them selectable with 'if else' statements using respective strings. However, AKAZE descriptor can only be used with KAZE or AKAZE keypoints, and the SIFT descriptor belong to HOG (Histogram or Oriented Gradient) family of descriptors, while BRIEF, ORB, FREAK, AKAZE are binary based descriptors. Also, unlike the other 3, BRIEF is a DescriptorExtractor, not a FeatureDetector.

MP.5. Descriptor Matching

As an alternative to the BF (Brut Force) matching, I implemented FLANN (Fast Library for Approximate Nearest Neighbors) library available in OpenCV using an 'if else' statement, which is an open-source library that builds a very efficient KD-tree data structure to search for matching pairs and avoids the exhaustive search of the brute force approach. Meanwhile, because of a potential bug in the current implementation of the OpenCV, it is required to convert the binary descriptors to floating point vectors as a workaround, which is still very inefficient.

Also, as an alternative to nearest neighbor selection, I also used an 'if else' statement to implement the k-nearest neighbor selection to finds the 2 best matches for each keypoint, with k = 2. Making both selectable using respective strings.

MP.6. Descriptor Distance Ratio

With the use of a 'for' loop I was able to filter matches using **descriptor distance ratio** of 0.8, to sort out ambiguous matches. In practice, a threshold value of 0.8 has proven to provide a good balance between TP (True Positives) and FP (False Positives).

Performance Evaluation

MP.7. Count the number of keypoints on the preceding vehicle for all 10 images and take note of the distribution of their neighborhood size. Do this for all the detectors you have implemented.

After getting the keypoints on the preceding vehicle in MP.3, I implemented the code to also print out the number of keypoints using the size() function i.e. keypoint.size(). This way I kept track of the number of keypoints detected by each of the detectors on every image, as we loop through them.

MP.8. Count the number of matched keypoints for all 10 images using all possible combinations of detectors and descriptors. In the matching step, the BF approach is used with the descriptor distance ratio set to 0.8.

The code is also implemented to prints out the number of matches made, the time it takes to make the matches in ms (milliseconds), and the number of matches removed after applying the descriptor distance ratio test. This is done for all possible combinations of detectors and descriptors using BF approach with the descriptor distance ratio set to 0.8 as instructed, but with an option to also use the FLANN approach.

MP.9.

I have included a spreadsheet where I logged the time it takes for keypoint detection and descriptor extraction for all ten images using all possible detector descriptor combinations. And based on the data in the spreadsheet, my top 3 detector descriptor combinations are:

1. FAST/ORB

2. FAST/BRIEF

3. ORB/ BRIEF

I recommend them as the best choice for our purpose of detecting keypoints on vehicles because, from the data, FAST/ORB combination is the fastest, followed closely by FAST/BRIEF, then comes ORB/ BRIEF which is not very close to the first two but still very fast compared to most other combinations. In addition, they also yielded very good number of keypoints, descriptors and as a result a very good number of matches when compared to the other detector descriptor combinations.

I also tried using the FLANN approach in the matching step, and the results turned out to very similar but with FAST/ORB and FAST/BRIEF switching positions i.e., FAST/BRIEF performed slightly better than FAST/ORB combination. While ORB/ BRIEF remains in third best position. This can also be seen in the spreadsheet (FAST/ORB and ORB/ BRIEF).

REFERENCE:

1. https://github.com/ragu-manjegowda/camera-based-feature-tracking/blob/master/src/matching2D_Student.cpp

2. https://github.com/eazydammy/camera-based-2d-feature-tracking/blob/main/src/matching2D_Student.cpp