Midterm report: Camera Based 2D Feature Tracking

Gia Minh Hoang

# M.P1 Data Buffer Optimization

In order to optimize image data loading, a ring buffer is implemented as shown in the following code. The buffer size is first compared with the maximum buffer size which is set to 2 for constant velocity model. If the buffer reaches its maximum capacity, the first (oldest) image in the buffer is erased. The new image is then loaded to the buffer.

if (dataBuffer.size() == dataBufferSize)

dataBuffer.erase(dataBuffer.begin());

dataBuffer.push\_back(frame);

On the other hand, the task can be done by queue in order not to shift all the elements in the buffer when it is full.

# MP.2 Keypoint Detection

The detectors e.g., SHITOMASI, HARRIS, FAST, BRISK, ORB, AKAZE, and SIFT are mainly implemented in matching2D\_Student.cpp from line 103 to line 259. The implementation of SHITOMASI detector is already provided by the instructor.

Lines 144 – 212 implements HARRIS detector using OpenCV through the command in line 156 i.e., cv::cornerHarris() to return detected keypoints stored in the matrix cv::Mat dst. Moreover, lines 161 – 196 preforms Non-Maximum Suppression (NMS) to filter to (i) ensure that the pixel with maximum cornerness in a local neighborhood and (ii) to prevent corners from being too close to each other. Particularly, new keypoint is checked if it overlaps the existing keypoints.

The FAST, BRISK, ORB, AKAZE, and SIFT are implemented in a single function detKeypointsModern (line 214) and are selected by the input string detectorType.

Lines 112 – 123 in file MidtermProject\_Camera\_Student.cpp perform the selection of the detector. The string DectectorType is used to call the corresponding detector mentioned above.

# MP.3 Keypoint Removal

Lines 131 – 148 remove the keypoints that are not in the subject of interest i.e., the preceding vehicle. It is done by comparing all the detected keypoints’s coordinates with a predefined rectangle bounding the preceding vehicle.

# MP.4 Keypoint Descriptors

Different descriptors such as BRISK, BRIEF, ORB, FREAK, AKAZE, and SIFT are implemented in matching2D\_Student.cpp from line 60 to line 100 through function descKeypoints(). The descriptor can be selected by the string descriptorType. All these descriptors are supported by OpenCV. The cv::DescriptorExtractor is created depending on the string descriptorType. In line 96, extractor->compute(img, keypoints, descriptors) performs feature description for the cv::Mat keypoints in the image cv::Mat img and stores the result in the cv::Mat descriptor.

# MP.5 Descriptor Matching

On the one hand, FLANN matching method is implemented in matching2D\_Student.cpp in lines 20 – 26. As guided, I have to convert binary descriptors to floating point due to a bug in current OpenCV implementation.

On the other hand, a keypoint may match with more than one other keypoint. Since the probability of selecting the wrong corresponding keypoint can be high, this matching should be rejected. To use this approach, the best two matching points of each keypoint are collected by using the knnMatch method of the cv::DescriptorMatcher class. Lines 38 – 43 in matching2D\_Student.cpp show how this strategy is done.

# MP.6 Descriptor Distance Ratio

Descriptor distance ratio is performed in matching2D\_Student.cpp lines 45 – 53 inside k nearest neighbors matching part. It rejects all the best matches with a matching distance similar to that of their second best match. Since knnMatch produces a std::vector of size k (k = 2 in this case), it can be done by looping over each keypoint match and performing a ratio test.

# MP.7 Performance Evaluation 1

As the keypoints are stored using the std::vector class. The number of keypoints on the preceding vehicle can be counted by simply using the container vector::size() e.g., keypoints.size() where vector<cv::KeyPoint> keypoints stores the detected keypoints.

# MP.8 Performance Evaluation 2

Similar to MP.7, the number of matched keypoints can be counted by matches.size() where vector<cv::DMatch> matches stores the matched keypoints.

# MP.9 Performance Evaluation 3

The execution time for keypoint detection and descriptor extraction is logged using all possible combinations (of detectors and descriptors). Since AKAZE descriptor only works with AKAZE detector and descriptor ORB cannot function with detector SIFT, we have 35 combinations.

The results is collected in the results.csv in directory /SFND\_2D\_Feature\_Tracking/report/

Based on the results, the top 3 detector/descriptor combinations are:

1. Detector FAST and descriptor BRIEF
2. Detector FAST and descriptor BRISK
3. Detector FAST and descriptor ORB

The FAST detector combining with the BRIEF, BRISK, and ORB descriptors are the fastest in execution time while maintaining rather good portion of keypoints on the preceding vehicle, and matching rather good portion of keypoints between successive images. As reaction time is critical for collision detection systems, the shorter execution time, the better in the condition that the number of keypoints is high enough in order to track the preceding vehicle.

In terms of the number of detected keypoints, BRISK detects the greatest number on the whole images and on the preceding vehicle but it is slower in execution time.

Overall, other combinations are either slower in execution time while giving a similar number of keypoints, or produce much more keypoints but are much slower execution time.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Detector | Descriptor | Image id | No keypoints | No keypoints on preceding vehicle | No matched keypoints | Keypoint detection time (ms) | Descriptor extraction time (ms) | Matching time (ms) |
| FAST | BRISK | 0 | 1824 | 149 | 0 | 2.7547 | 2.2352 | 0 |
| FAST | BRISK | 1 | 1832 | 152 | 97 | 1.8401 | 2.0678 | 0.3212 |
| FAST | BRISK | 2 | 1810 | 152 | 103 | 1.6613 | 2.0679 | 0.2676 |
| FAST | BRISK | 3 | 1817 | 157 | 104 | 1.6015 | 2.0778 | 0.279 |
| FAST | BRISK | 4 | 1793 | 149 | 98 | 1.6335 | 1.9502 | 0.2758 |
| FAST | BRISK | 5 | 1796 | 150 | 86 | 1.5327 | 3.0986 | 0.27 |
| FAST | BRISK | 6 | 1788 | 157 | 108 | 1.5603 | 2.2184 | 0.2435 |
| FAST | BRISK | 7 | 1695 | 152 | 108 | 1.4423 | 1.9763 | 0.3376 |
| FAST | BRISK | 8 | 1749 | 139 | 100 | 1.4621 | 1.7752 | 0.2622 |
| FAST | BRISK | 9 | 1770 | 144 | 100 | 1.4032 | 2.2863 | 0.2684 |
| FAST | BRIEF | 0 | 1824 | 149 | 0 | 1.7432 | 0.8565 | 0 |
| FAST | BRIEF | 1 | 1832 | 152 | 119 | 1.7821 | 1.05 | 0.5901 |
| FAST | BRIEF | 2 | 1810 | 152 | 129 | 1.8879 | 1.276 | 0.3078 |
| FAST | BRIEF | 3 | 1817 | 157 | 122 | 1.7757 | 1.0179 | 0.3134 |
| FAST | BRIEF | 4 | 1793 | 149 | 126 | 2.4953 | 0.9087 | 0.3773 |
| FAST | BRIEF | 5 | 1796 | 150 | 109 | 1.7391 | 1.3574 | 0.2889 |
| FAST | BRIEF | 6 | 1788 | 157 | 124 | 1.9265 | 0.9114 | 0.4322 |
| FAST | BRIEF | 7 | 1695 | 152 | 132 | 1.805 | 0.9464 | 0.2964 |
| FAST | BRIEF | 8 | 1749 | 139 | 125 | 1.8519 | 0.847 | 0.2847 |
| FAST | BRIEF | 9 | 1770 | 144 | 120 | 2.2312 | 1.1137 | 0.2651 |
| FAST | ORB | 0 | 1824 | 149 | 0 | 1.9892 | 5.1932 | 0 |
| FAST | ORB | 1 | 1832 | 152 | 122 | 1.8614 | 5.1772 | 0.2969 |
| FAST | ORB | 2 | 1810 | 152 | 123 | 1.8587 | 5.0772 | 0.3131 |
| FAST | ORB | 3 | 1817 | 157 | 119 | 1.8697 | 6.0033 | 0.3057 |
| FAST | ORB | 4 | 1793 | 149 | 129 | 1.8913 | 5.1322 | 0.3359 |
| FAST | ORB | 5 | 1796 | 150 | 108 | 1.7312 | 4.8355 | 0.2668 |
| FAST | ORB | 6 | 1788 | 157 | 121 | 2.4802 | 5.1116 | 0.2804 |
| FAST | ORB | 7 | 1695 | 152 | 127 | 1.8262 | 5.0757 | 0.2812 |
| FAST | ORB | 8 | 1749 | 139 | 122 | 1.8499 | 5.9252 | 0.3747 |
| FAST | ORB | 9 | 1770 | 144 | 119 | 1.835 | 4.9868 | 0.2764 |