Mid Term Project Report **Ayush Jain**
Camera Lessons Submission Date: 27/04/2020
Sensor Fusion Nano degree
Udacity

MP.1 Data Buffer Optimization

Q. Implement a vector for dataBuffer objects whose size does not exceed a limit (e.g. 2 elements). This can be achieved by pushing in new elements on one end and removing elements on the other end.
Ans. In order to retain the ring buffer of size **dataBufferSize**, an **if** condition is used on line 151 of **MidTermProject_Camera_Student.cpp**. The **if** condition checks to see if the **dataBuffer** is equal to the **dataBufferSize**, and if so, the front of the vector is removed using the **erase** method, and the next frame is pushed to the back. Thus the overloading of the memory was avoided.

```
1 if (dataBuffer.size() == dataBufferSize) { dataBuffer.erase(std::begin(
    dataBuffer)); }
2
3 dataBuffer.push_back(frame);
```

In order to access different keypoint detectors, if else loop have been defined in detKeypointsModern function in matching2D_student.cpp. All the keypoint detection methods such as Classical -¿ HARRIS and the Modern -¿ FAST, BRISK, ORB, AKAZE, SIFT are accessible.

MP.2 Keypoint Detection

Q. Implement detectors HARRIS, FAST, BRISK, ORB, AKAZE, and SIFT and make them selectable by setting a string accordingly.
Ans. Within an **if**, **else if**, and **else** conditions, the **Classical: Shi-Tomasi, Harris,** and the **Modern: FAST, BRISK, ORB, AKAZE,** and **SIFT** detectors are executed. On line 169 of **MidTermProject_Camera_Student.cpp**, the **detectorType** is compared against the string of **HARRIS**, and if the strings are equal, the **detKeypointsHarris** function is called. Otherwise, the **detKeypointsModern** function is executed, which contains a series of **if** conditions that compare the **detectorType** string against other string literals.

Within **detKeypointsModern**, a **cv::Ptr** of **FeatureDetector** type is assigned the proper detector based on the chosen detector. Once assigned, the **detector:detect** method is executed, which takes the grayscale image as input, and stores the detected keypoints in the **keypoints** vector. The results are displayed if the user has chosen to visualize them.

The number of keypoints detected and the elapsed time is returned in a struct, which is used to populate a vector, and in-turn fill out the final **CSV** file. This pattern is repeated with all function calls in the **matching2D_Student.cpp** file.

MP.3 Keypoint Removal

Q. Remove all keypoints outside of a pre-defined rectangle and only use the keypoints within the rectangle for further processing.
Ans. The keypoints found within the bounding box of the preceding vehicle are extracted within the **if** condition on line 189 of **MidTermProject_Camera_Student.cpp**. An auxilary vector named **retainedPoints** is used to store all the points found within the bounding box named

vehicleRect. For each point in the **keypoints** vector, the **vehicleRect.contains** method is called to determine if that point exists within the boundary. If the condition is true, that point is copied to the **retainedPoints** vector. When the for-loop finishes, the **keypoints** vector is assigned the contents of the **retainedPoints** vector.

MP.4 Keypoint Descriptors

Q. Implement descriptors BRIEF, ORB, FREAK, AKAZE and SIFT and make them selectable by setting a string accordingly.
Ans. The **BRIEF, ORB, FREAK, AKAZE,** and **SIFT** descriptors were implemented in a manner similar to the detectors from **MP.2**. Beginning on line 80 of **matching2D_Student.cpp**, a series of **if** conditions are used to compare the string input by the user. If the string matches one of the hard-coded descriptor types, the generic **extractor** variable is assigned the appropriate descriptor, and the **compute** method is called. Additionally, the elapsed time is measured during execution.

MP.5 Descriptor Matching

Q. Implement FLANN matching as well as k-nearest neighbor selection. Both methods must be selectable using the respective strings in the main function.
Ans. FLANN matching was implemented on line 27 of **matching2D_Student.cpp**. The **matcher cv:Ptr** of **cv::DescriptorMatcher** type is assigned a pointer to a descriptor matcher constructed with a **FLANNBASED** type. Additionally, two **if** conditions are used to determine if the descriptor matrices are not **CV_32F** type, and if so, they are converted in order to avoid an existing bug in OpenCV.

MP.6 Descriptor Distance Ratio

Q. Use the K-Nearest-Neighbor matching to implement the descriptor distance ratio test, which looks at the ratio of best vs. second-best match to decide whether to keep an associated pair of keypoints.
Ans. K-Nearest-Neighbor selection is implemented in **matching2D_Student.cpp** beginning on line 48. A 2D vector of **cv::DMatch** type is used to store the matches from calling **matcher:knnMatch**, using a value of 2 for k. Next, for each match in the **knnMatches** vector, the descriptor distance ratio test is performed. The distance threshold is set to 0.8, and each point falling within the threshold distance is copied to the **matches** vector.

MP.7 Performance Evaluation 1

Q. 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.
Ans. The number of keypoints on the preceding vehicle are recorded within the if condition beginning on line 189. This task is accomplished as side-effect of task **MP.3**, where the number of keypoints found on the preceding vehicle is equal to the size of **retainedPoints** vector. The results are stored in the output CSV file.

MP.8 Performance Evaluation 2

Q. 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.
Ans. The number of matched keypoints are determined in the **matchDescriptors** function in **matching2D_Student.cpp**. This is again accomplished as a side-effect of task **MP.6**, and the number of matched points is equal to the size of the matches vector. The results are stored in the output CSV file.

MP.9 Performance Evaluation 3

Q. Log the time it takes for keypoint detection and descriptor extraction. The results must be entered into a spreadsheet and based on this data, the TOP3 detector / descriptor combinations must be recommended as the best choice for our purpose of detecting keypoints on vehicles.
Ans. The final results can be found in the CSV file named **Ayush_Jain_Midterm_Project.csv** within the report directory. Based on the final results, the top 3 detector/descriptor combinations for this project are:

1. Detector: FAST, Descriptor: BRIEF
2. Detector: FAST, Descriptor: ORB
3. Detector: FAST, Descriptor: BRISK

The **FAST** detector in combination with the **BRIEF, ORB,** and **BRISK** descriptors executed in the shortest amount of time. Additionaly, they were able to maintain a good portion of points on the preceding vehicle, and match a good portion of points between successive images. These three combinations retained much of the detail required to detect and track vehicles on the road. Reaction time for autonomous vehicles is extremely critical, thus having a performant dectector and descriptor combination is of utmost importance.

The **BRISK** detector was able to detect more initial points, and retain more points on the preceding vehicle, but with significantly slower execution time. In general, all other detector/descriptor combinations either executed more slowly and produced a similiar number of keypoints, or produced a larger number of keypoints, but with an even extremely slow execution time.

Note: The execution times shown in this screenshot are from running the program on my desktop computer. The timing data may vary when running in the student workspace.

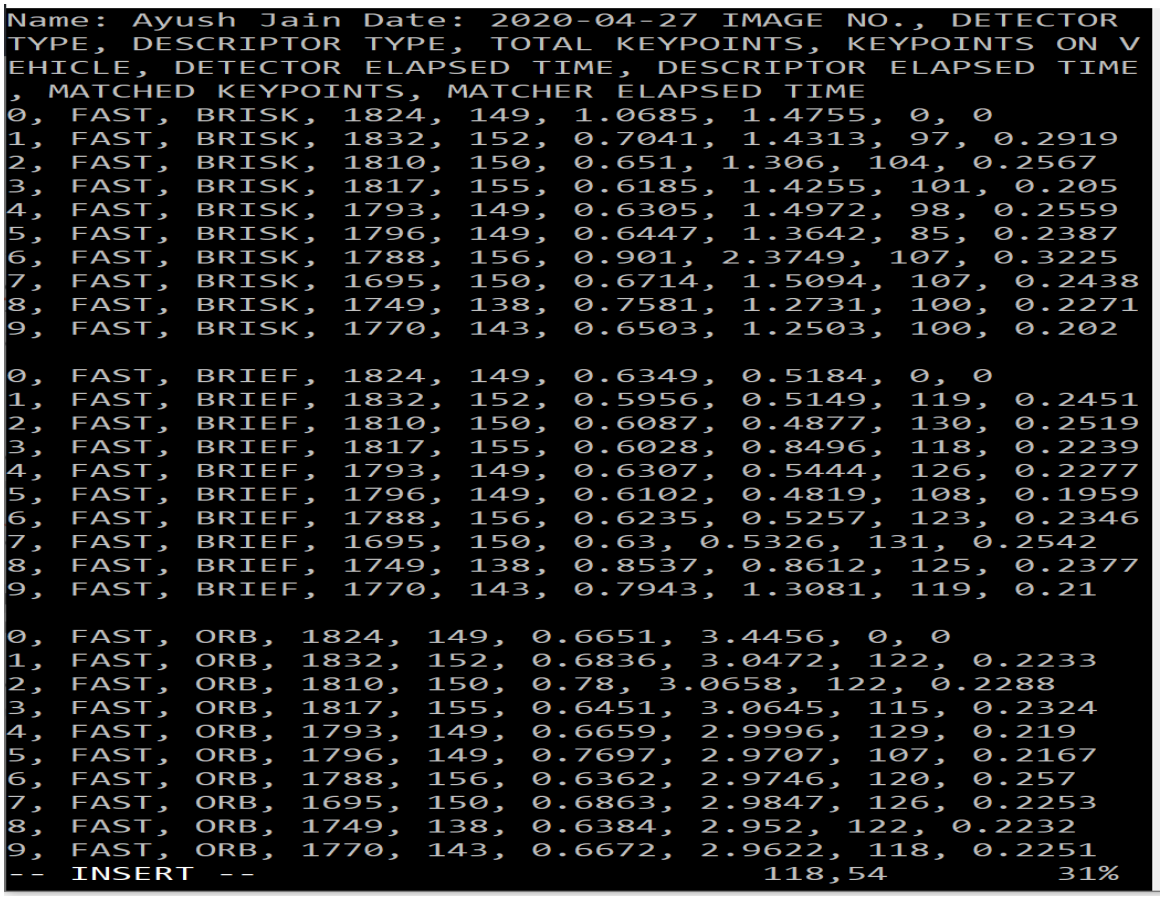

Figure 1: Figure1: