Camera Based 2D Feature Tracking Mid-term ReadMe

Arjun Agrawal

5th July 2021

MP.1 Data Buffer Optimization:
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.

```
dataBuffer.push_back(frame);
if(dataBuffer.size()>dataBufferSize){
    dataBuffer.erase(dataBuffer.begin());
}
```

In order to maintain a constant buffer size, whenever a new piece of data is added the oldest piece of data is erased.

MP.2 Keypoint Detection:
Implement detectors HARRIS, FAST, BRISK, ORB, AKAZE, and SIFT and make them selectable by setting a string accordingly.

if-else branches were implemented to allow for the selection of various detectors. The work done in the previous exercises was used as a guide to deploy all of the detectors in the matching2D_Student.cpp.

MP.3 Keypoint Removal:
Remove all keypoints outside of a predefined rectangle and only use the keypoints within the rectangle for further processing.

```
cv::Rect vehicleRect(535, 180, 180, 150);
if (bFocusOnVehicle) {
  for (auto it = keypoints.begin(); it != keypoints.end(); it++)
    if( !vehicleRect.contains(it->pt)) {
      keypoints.erase(it);
      --it;
    }
}
```

The points in the keypoints vector are checked with reference to a rectangle outlining the vehicle in front of the ego vehicle. This is computed with the help of the `contains` function.

MP.4 Keypoint Descriptors:
Implement descriptors BRIEF, ORB, FREAK, AKAZE and SIFT and make them selectable by setting a string accordingly.

A if-else branch was created within the `descKeypoints` function to tackle this task. A warning is created for incorrect string input.

MP.5 Descriptor Matching:
Implement FLANN matching as well as k-nearest neighbor selection. Both methods must be selectable using the respective strings in the main function.

The implementation of FLANN was carried out with the help of the `DescriptorMatcher::FLANNBASED`. In order for it to work, I had to ensure that the descriptor data was of type single-precision (32-bit) floating point.

The `knnMatch` method is used to address the k-nearest neighbor selection. Along with knn, cross checking cannot be used. Hence I also had to ensure that cross checking was turned off.

MP.6 Descriptor Distance Ratio:
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.

```
double minDescDistRatio = 0.8;
for (auto it = knn_matches.begin(); it != knn_matches.end(); ++it)
{
    if ((*it)[0].distance < minDescDistRatio * (*it)[1].distance)
    {
        matches.push_back((*it)[0]);
    }
}
```

The descriptor distance ratio is implemented in order to only select a match if the distance ratio is less than 0.8.

MP.7 Performance Evaluation 1:
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.

A vector was used to store the number of keypoints detected at each image within the rectangle. These values were summed to provide the total number of keypoints detected by each detector.

No. of keypoints detected by each detector			
		Total no. of keypoints	Average no. of keypoints
DETECTOR	SHITOMASI	1179	117.9
	HARRIS	248	24.8
	FAST	1491	149.1
	BRISK	2762	276.2
	ORB	1161	116.1
	AKAZE	1670	167
	SIFT	1386	138.6

MP.8 Performance Evaluation 2:
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.

In order to make the data collection process less time consuming a shell script Eval.sh. Note that AKAZE descriptors only work with AKAZE keypoints while all descriptors can use the AKAZE keypoints. There is also an error raised by openCV when SIFT keypoints are provided to the ORB descriptor.

Total no. of matched keypoints across all 10 images							
		DESCRIPTOR					
		BRISK	BRIEF	ORB	FREAK	AKAZE	SIFT
DETECTOR	SHITOMASI	797	963	925	797	N/A	944
	HARRIS	142	173	160	146	N/A	163
	FAST	963	1140	1123	942	N/A	1081
	BRISK	1676	1759	1621	1526	N/A	1735
	ORB	781	545	761	421	N/A	763
	AKAZE	1215	1266	1186	1188	1259	1270
	SIFT	592	702	N/A	596	N/A	800

MP.9 Performance Evaluation 3:
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.

My criteria for selecting a suitable for our purpose in autonomous automotive navigation places a priority on computational speed. This is because we need the overall system to react quickly to changes in the environment in order to maintain vehicle safety and efficiency in path planning. The table below shows us that the FAST detector leads to the fastest overall time. Hence, the following are my top three detector / descriptor combinations.

1. FAST detector with BRIEF keypoints
2. FAST detector with BRISK keypoints
3. FAST detector with ORB keypoints

Though in reality, we would have to check the quality of the key points detected. We need a way to ensure that the key points generated by the FAST detector lead to a high True Positive Rate and Precision while maintaining a low False Positive Rate.

Total time taken for keypoint detection and descriptor computation across all 10 images							
		DESCRIPTOR					
		BRISK	BRIEF	ORB	FREAK	AKAZE	SIFT
DETECTOR	SHITOMASI	545.015	453.158	535.375	1606.09	N/A	1000.74
	HARRIS	591.84	576.049	632.368	1543.24	N/A	1150.68
	FAST	122.621	71.5719	145.319	1171.03	N/A	893.697
	BRISK	2048.82	2129.76	2397.91	3128.72	N/A	3568.59
	ORB	702.517	548.24	887.921	1535.8	N/A	2081.97
	AKAZE	3857.72	3608.57	3821.64	4854.34	6370.31	4766.42
	SIFT	5686.5	4616.62	N/A	5693.62	N/A	7246.53