1. 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.

MidTermProject_Camera_Student.cpp

```
// push image into data frame buffer
DataFrame frame;
frame.cameraImg = imgGray;
if (dataBuffer.size()>=dataBufferSize)
{
        dataBuffer.erase(dataBuffer.begin());
}
dataBuffer.push_back(frame);
```

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

MidTermProject_Camera_Student.cpp

```
if (detectorType.compare("SHITOMASI") == 0)
{
        detKeypointsShiTomasi(keypoints, imgGray, false);
}

else if (detectorType.compare("HARRIS") == 0)
{
        detKeypointsHarris(keypoints, imgGray, false);
}

else
{
        detKeypointsModern(keypoints, imgGray, detectorType, false);
}
```

matching2D_Student.cpp

```
void detKeypointsHarris(std::vector<cv::KeyPoint> &keypoints, cv::Mat &img, bool bVis)
```

```
void detKeypointsShiTomasi(vector<cv::KeyPoint> &keypoints, cv::Mat &img, bool bVis)
```

```
void detKeypointsModern(vector<cv::KeyPoint> &keypoints, cv::Mat &img, string detectorType, bool bVis)
```

3. Remove all keypoints outside of a pre-defined rectangle and only use the keypoints within the rectangle for further processing.

MidTermProject_Camera_Student.cpp

```
// only keep keypoints on the preceding vehicle
bool bFocusOnVehicle = true;
cv::Rect vehicleRect(535, 180, 180, 150);
vector<cv::KeyPoint>::iterator keypoint;
vector<cv::KeyPoint> keypoints_roi;
if (bFocusOnVehicle)
{
        for(keypoint = keypoints.begin(); keypoint != keypoints.end(); ++keypoint)
        {
                if (vehicleRect.contains(keypoint->pt))
                {
                        cv::KeyPoint newKeyPoint;
                        newKeyPoint.pt = cv::Point2f(keypoint->pt);
                        newKeyPoint.size = 1;
                        keypoints_roi.push_back(newKeyPoint);
                }
        }
       keypoints =  keypoints_roi;
       cout << "IN ROI n= " << keypoints.size()<<" keypoints"<<endl;
}
```

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

MidTermProject_Camera_Student.cpp

```
cv::Mat descriptors;
string descriptorType = "BRISK"; // BRISK, BRIEF, ORB, FREAK, AKAZE, SIFT
descKeypoints((dataBuffer.end() - 1)->keypoints, (dataBuffer.end() - 1)->cameraImg, descriptors, descriptorType);
```

matching2D_Student.cpp

```
53  void descKeypoints(vector<cv::KeyPoint> &keypoints, cv::Mat &img, cv::Mat &descriptors, string descriptorType)
54  {
55          // select appropriate descriptor
56          cv::Ptr<cv::DescriptorExtractor> extractor;
57          if (descriptorType.compare("BRISK") == 0)
58          {
59
60                  int threshold = 30;               // FAST/AGAST detection threshold score.
61                  int octaves = 3;                  // detection octaves (use 0 to do single scale)
62                  float patternScale = 1.0f; // apply this scale to the pattern used for sampling the neighbourhood
63
64                  extractor = cv::BRISK::create(threshold, octaves, patternScale);
65          }
66          else if (descriptorType.compare("BRIEF") == 0)
67          {
68                  extractor = cv::xfeatures2d::BriefDescriptorExtractor::create();
69          }
70          else if (descriptorType.compare("ORB") == 0)
71          {
72                  extractor = cv::ORB::create();
73          }
74          else if (descriptorType.compare("FREAK") == 0)
75          {
76                  extractor = cv::xfeatures2d::FREAK::create();
77          }
78          else if (descriptorType.compare("AKAZE") == 0)
79          {
80                  extractor = cv::AKAZE::create();
81          }
82          else if (descriptorType.compare("SIFT") == 0)
83          {
84                  extractor = cv::xfeatures2d::SIFT::create();
85          }
86
87          // perform feature description
88          double t = (double)cv::getTickCount();
89          extractor->compute(img, keypoints, descriptors);
90          t = ((double)cv::getTickCount() - t) / cv::getTickFrequency();
91          cout << descriptorType << " descriptor extraction in " << 1000 * t / 1.0 << " ms" << endl;
92  }
```

5. Implement FLANN matching as well as k-nearest neighbor selection. Both methods must be

selectable using the respective strings in the main function.

matching2D_Student.cpp

```
14        if (matcherType.compare("MAT_BF") == 0)
15        {
16            int normType = cv::NORM_HAMMING;
17            matcher = cv::BFMatcher::create(normType, crossCheck);
18        }
19        else if (matcherType.compare("MAT_FLANN") == 0)
20        {
21            if (descSource.type() != CV_32F)
22            {
23                descSource.convertTo(descSource, CV_32F);
24                descRef.convertTo(descRef, CV_32F);
25            }
26
27            matcher = cv::DescriptorMatcher::create(cv::DescriptorMatcher::FLANNBASED);
28        }
```

6. 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.

matching2D_Student.cpp

```
30        // perform matching task
31        if (selectorType.compare("SEL_NN") == 0)
32        { // nearest neighbor (best match)
33            matcher->match(descSource, descRef, matches); // Finds the best match for each descriptor in desc1
34        }
35        else if (selectorType.compare("SEL_KNN") == 0)
36        { // k nearest neighbors (k=2)
37
38            vector<vector<cv::DMatch>> knn_matches;
39            matcher ->knnMatch(descSource, descRef, knn_matches, 2);
40
41            double minDescDistratio = 0.8;
42            for (auto it = knn_matches.begin(); it != knn_matches.end(); ++it)
43            {
44                if ((*it)[0].distance<minDescDistratio * (*it)[1].distance)
45                {
46                    matches.push_back((*it)[0]);
47                }
48            }
49        }
```

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

| Detector | Number of detected Keypoints on the preceding vehicle for total of 10 images |
|---|---|
| SHITOMASI | 1179 |
| HARRIS | 560 |
| FAST | 1491 |
| BRISK | 2762 |
| ORB | 1161 |
| AKAZE | 1670 |
| SIFT | 1386 |

8. 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.

| Detector\Descriptor | BRISK | BRIEF | ORB | FREAK | AKAZE | SIFT |
|---|---|---|---|---|---|---|
| SHITOMASI | 767 | 944 | 908 | 768 | N/A | 927 |
| HARRIS | 393 | 460 | 451 | 396 | N/A | 459 |
| FAST | 899 | 1099 | 1071 | 878 | N/A | 1046 |

| | | | | | | |
|---|---|---|---|---|---|---|
| BRISK | 1570 | 1704 | 1514 | 1524 | N/A | 1646 |
| ORB | 751 | 545 | 763 | 420 | N/A | 763 |
| AKAZE | 1215 | 1266 | 1182 | 1187 | 1259 | 1270 |
| SIFT | 592 | 702 | Out of Memory | 593 | N/A | 800 |

9. Average Processing Time (ms) on all images for each detector/descriptor combination

| Detector\Descriptor | BRISK | BRIEF | ORB | FREAK | AKAZE | SIFT |
|---|---|---|---|---|---|---|
| SHITOMASI | 17.98 | 21.38 | 18.8 | 52.4079 | N/A | 31.82 |
| HARRIS | 16.98 | 17.11 | 16.5383 | 51.01 | N/A | 32.73 |
| FAST | 3.36 | 1.8786 | 2.03823 | 41.549 | N/A | 35.71 |
| BRISK | 43.736 | 44.159 | 47.966 | 89.2 | N/A | 92.17 |
| ORB | 8.54 | 8.035 | 11.81 | 47.578 | N/A | 51.6748 |
| AKAZE | 103.108 | 81.3247 | 84.51 | 149.97 | 173.611 | 128.77 |
| SIFT | 124.09 | 146.49 | Out of Memory | 188.17 | N/A | 181.0381 |

Based on the results above, the top 3 detector/descriptor combinations, that achieve minimal processing time with significant matches are

| DETECTOR/DESCRIPTOR | NUMBER OF KEYPOINTS | TIME |
|---|---|---|
| FAST+BRIEF | 1099 keypoints | 1,87 ms |
| FAST+ORB | 1071 keypoints | 2.03 ms |
| FAST+BRISK | 899 keypoints | 3.36 ms |