TASK MP.1 (Data Buffer)

For implementing the ring buffer, the following code remove the old image from one end of the buffer when it gets full and then a new image could be pushed to the buffer on the other end
File : MidTermProject_Camera_Student.cpp

```
        //// STUDENT ASSIGNMENT
        //// TASK MP.1 -> replace the following code with ring buffer of
size dataBufferSize

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

        //// EOF STUDENT ASSIGNMENT
```

TASK MP.2 (Keypoint Detection)

Implementing the different types of detectors which are selectable by setting the string detectorType. The different types of detectors are implemented as follows:
File: MidTermProject_Camera_Student.cpp

```
//// STUDENT ASSIGNMENT
        //// TASK MP.2 -> add the following keypoint detectors in file
matching2D.cpp and enable string-based selection based on detectorType
        //// -> HARRIS, FAST, BRISK, ORB, AKAZE, SIFT
        t = (double)cv::getTickCount();
        bool bVis = false;
        if (detectorType.compare("SHITOMASI") == 0)
        {
            detKeypointsShiTomasi(keypoints, imgGray, bVis);
        }
        else if (detectorType.compare("HARRIS") == 0)
        {
            detKeypointsHarris(keypoints, imgGray, bVis);
        }
        else
        {
            detKeypointsModern(keypoints, imgGray, detectorType, bVis);
        }
        //// EOF STUDENT ASSIGNMENT
```

The function detKeypointsModern implements all the modern detectors like (`"FAST" "BRISK" "ORB" "AKAZE"`) in the file matching2D_Student.cpp.

TASK MP.3 Keypoint Removal

In this task I considered only the keypoints which are inside a predefined rectangle as follows:

```
//// STUDENT ASSIGNMENT
        //// TASK MP.3 -> only keep keypoints on the preceding vehicle

        // only keep keypoints on the preceding vehicle
        bool bFocusOnVehicle = true;
        cv::Rect vehicleRect(535, 180, 180, 150);
        if (bFocusOnVehicle)
        {
            double kptsTotal = keypoints.size();
            vector<cv::KeyPoint> keypointsFocus;
            for(auto it=keypoints.begin();it!=keypoints.end();++it)
            {
                if(vehicleRect.contains((*it).pt) )
                {
                    cv::KeyPoint newKeyPoint;
                    newKeyPoint.pt = cv::Point2f((*it).pt);
                    newKeyPoint.size = 1;
                    keypointsFocus.push_back(newKeyPoint);
                }
            }
            keypoints = keypointsFocus;
```

TASK MP.4 Keypoint Descriptors

Implementing the different types of descriptors which are selectable by setting the string descriptorType. The different types of descriptors are implemented as follows:
File: MidTermProject_Camera_Student.cpp

```
//// STUDENT ASSIGNMENT
        //// TASK MP.4 -> add the following descriptors in file matching2D.cpp and enable string-based selection based on descriptorType
        //// -> BRIEF, ORB, FREAK, AKAZE, SIFT

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

The descKeypoints function is implemented in the file Matching2D_Student.cpp as follows:

```
cv::Ptr<cv::DescriptorExtractor> extractor;
    if (descriptorType.compare("BRISK") == 0)
    {

        int threshold = 30;        // FAST/AGAST detection threshold score.
        int octaves = 3;           // detection octaves (use 0 to do single scale)
        float patternScale = 1.0f; // apply this scale to the pattern used for sampling the neighbourhood of a keypoint.

        extractor = cv::BRISK::create(threshold, octaves, patternScale);
    }
    else if(descriptorType.compare("BRIEF") == 0)
    {
        extractor = cv::xfeatures2d::BriefDescriptorExtractor::create();
    }
    else if(descriptorType.compare("ORB") == 0)
    {
        extractor = cv::ORB::create();
    }
    else if(descriptorType.compare("FREAK") == 0)
    {
        extractor = cv::xfeatures2d::FREAK::create();
    }
    else if(descriptorType.compare("AKAZE") == 0)
    {
        extractor = cv::AKAZE::create();
    }
    else if(descriptorType.compare("SIFT") == 0)
    {
        //extractor = cv::xfeatures2d::SIFT::create();
    }
    else
    {
        cout << "Descriptor type is invalid or out of scope" << endl;
    }


    // perform feature description
    double t = (double)cv::getTickCount();
    extractor->compute(img, keypoints, descriptors);
```

TASK MP.5 / 6 Descriptor Matching and Descriptor Distance Ratio

The implementation of FLANN matching with KNN selection including the descriptor distance ratio of 0.8 is called here as follows:

```
            //// STUDENT ASSIGNMENT
            //// TASK MP.5 -> add FLANN matching in file matching2D.cpp
            //// TASK MP.6 -> add KNN match selection and perform 
descriptor distance ratio filtering with t=0.8 in file matching2D.cpp

            matchDescriptors((dataBuffer.end() - 2)->keypoints, 
(dataBuffer.end() - 1)->keypoints,
                            (dataBuffer.end() - 2)->descriptors,
(dataBuffer.end() - 1)->descriptors,
                            matches, descriptorType, matcherType,
selectorType);

            //// EOF STUDENT ASSIGNMENT
```

The implementation of matchDescriptors function is as follows:
File Matching2D_Student.cpp

```
if (matcherType.compare("MAT_BF") == 0)
    {
        int normType = cv::NORM_HAMMING;
        matcher = cv::BFMatcher::create(normType, crossCheck);
    }
    else if (matcherType.compare("MAT_FLANN") == 0)
    {
        if (descSource.type() != CV_32F)
        { // OpenCV bug workaround : convert binary descriptors to floating
point due to a bug in current OpenCV implementation
            descSource.convertTo(descSource, CV_32F);
            descRef.convertTo(descRef, CV_32F);
        }

        matcher =
cv::DescriptorMatcher::create(cv::DescriptorMatcher::FLANNBASED);
    }

    // perform matching task
    if (selectorType.compare("SEL_NN") == 0)
    { // nearest neighbor (best match)

        matcher->match(descSource, descRef, matches); // Finds the best
match for each descriptor in desc1
    }
    else if (selectorType.compare("SEL_KNN") == 0)
    { // k nearest neighbors (k=2)

```

```
        vector<vector<cv::DMatch>> knn_matches;

        matcher->knnMatch(descSource, descRef, knn_matches, 2); // finds the 2 best matches

        // filter matches using descriptor distance ratio test
        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]);
            }
        }
    }
```

TASK MP.7 Performance Evaluation 1

I used the vector vkptsFocus to store the keypoints number for the 10 images as follows:

```
if (bFocusOnVehicle)
        {
            double kptsTotal = keypoints.size();
            vector<cv::KeyPoint> keypointsFocus;
            for(auto it=keypoints.begin();it!=keypoints.end();++it)
            {
                if(vehicleRect.contains((*it).pt) )
                {
                    cv::KeyPoint newKeyPoint;
                    newKeyPoint.pt = cv::Point2f((*it).pt);
                    newKeyPoint.size = 1;
                    keypointsFocus.push_back(newKeyPoint);
                }
            }
            keypoints = keypointsFocus;
            double kptsFocus = keypoints.size();
            cout<< "TASK MP.7: Original Keypoints: " << kptsTotal << " and
after focusing on Preceding vehicle became " << kptsFocus << " Keypoints"
<< endl;
            vkptsFocus.push_back(kptsFocus);
        }
```

The result after running the code with the different detectors as follows:

Detector / Image no.	1	2	3	4	5	6	7	8	9	10
SHITOMASI	125	118	123	120	120	113	114	123	111	112
HARRIS	17	14	18	21	26	43	18	31	26	34
FAST	149	152	150	155	149	149	156	150	138	143
BRISK	264	282	282	277	297	279	289	272	266	254
ORB	92	102	106	113	109	125	130	129	127	128
AKAZE	166	157	161	155	163	164	173	175	177	179

TASK MP.8,9 Performance Evaluation 2,3

I made a bash script to call the ./2D_feature_tracking executable with all the combination of detectors and descriptors as follows:

```
cd build
declare -a detector=("SHITOMASI" "HARRIS" "FAST" "BRISK" "ORB" "AKAZE")
declare -a descriptor=("BRISK" "BRIEF" "ORB" "FREAK" "AKAZE")
for i in "${detector[@]}"
do
  for j in "${descriptor[@]}"
  do
     ./2D_feature_tracking "$i" "$j"
  done
done
```

The results of the matched keypoints number and time taken for each detector/descriptor compination as well as an average for the matched keypoints and avg time for each combination are written into a file as follows:

```
double avgMatchedKpts = 0;
   double avgTime = 0;
   ofstream outfile;
   outfile.open("../perfEvaluation.txt", fstream::app);
   outfile << "Detector Type: " << detectorType << " Descriptor Type: " <<
descriptorType << endl << endl;
    for(int i = 0; i< vkptsMatched.size();i++)
    {
        outfile << "TASK.8.9: Image Number :" << i+1 << " has ," <<
vkptsMatched[i] << ", MatchedKeypoints" << "and took ," << 1000 *
timeDetDesc[i] / 1 << ", ms" << endl;
        avgMatchedKpts += vkptsMatched[i];
        avgTime += (1000 * timeDetDesc[i]) / 1;
    }
    outfile << "Average Time: " << avgTime / timeDetDesc.size() << "
Average Matched Keypoints: " << avgMatchedKpts / vkptsMatched.size() <<
endl;
    outfile << endl;
   outfile.close();

   return 0;
```

Then I created a spreadsheet with the previous info which could be found in the following path: SFND_2D_Feature_Tracking_Student/Camera_Based_2D_Feature_Tracking_perfEvaluation.xlsx first sheet

The Top3 detector/descriptor combination which have really reduced avg processing time and relatively small average matched keypoints are as follows

Top 3 Detector / Descriptor	Average Time	Average Matched Keypoints
FAST / ORB	2.56128	122.111
FAST / BRIEF	7.43224	120.111
ORB / BRIEF	16.9135	60.5556

NOTE1: In all the previous results SIFT is not considered and it was commented in my code since I faced so many problems in compilation and include xfeatures2d module and it keeps not working with me that’s why I continued without this type and I was planning to test it when I

upload the code to the udacity workspace and the results will be given at perfEvaluationSift.txt file.

NOTE2: any detector other than AKAZE type didn’t work with the descriptor of the AKAZE type.