Camera 2D Feature Matching – Sensor Fusion Nanodegree

MP.1 Data Buffer Optimization:

The implementation was done by pushing the new image at the end of the buffer and if buffer is full, the image at the top is removed.

```
// push image into data frame buffer
DataFrame frame;
frame.cameraImg = imgGray;
dataBuffer.push_back(frame);

// if the buffer size exceeds; remove the image from the start of the buffer
if(dataBuffer.size() > dataBufferSize )
{
    dataBuffer.erase(dataBuffer.begin());
}
```

MP.2 Keypoint Detection:

All detectors HARRIS, FAST, BRISK, ORB, AKAZE, and SIFT are implemented in the matching2D_Student.cpp file. The detectors can be selected using their names (datatype: string).

HARRIS detector is implemented in the function **detKeypointsHarris,** and all other modern detectors are implemented in the function **detKeypointsModern.**

If user enters an invalid detector type, BRISK detector will be executed by default.

```
if (detectorType.compare("SHITOMASI") == 0)
{
    detectTime = detKeypointsShiTomasi(keypoints, imgGray, false);
}
else if(detectorType.compare("HARRIS") == 0)
{
    detectTime = detKeypointsHarris(keypoints, imgGray, false);
}
else
{
    detectTime = detKeypointsModern(keypoints, imgGray, detectorType, false);
}
```

MP.3 Keypoint Removal:

AS the focus of project is only to find the TTC, we will concentrate on key points only in the vehicle which is travelling ahead. Removal of key points outside the vehicle ahead is done by defining a rectangle and removing points outside this rectangle.

```
// only keep keypoints on the preceding vehicle
bool bFocusOnVehicle = true;
cv::Rect vehicleRect(535, 180, 180, 150);
if (bFocusOnVehicle)
{
    for(auto keypt_it = keypoints.begin(); keypt_it != keypoints.end(); )
    {
        if(vehicleRect.contains(keypt_it->pt) == false)  //if keypoint not inside vehicle delete it
        {
            keypoints.erase(keypt_it);
        }
        else
        {
            keypt_it++;  // move to next key point if current key point is inside vehicle
        }
    }

    std::cout<<detectorType<<" : Keypoints inside the vehicle ahead : "<<keypoints.size()<<std::endl;
}
```

MP.4 Keypoint Descriptors

All descriptor BRIEF, ORB, FREAK, AKAZE and SIFT are implemented in the matching2D_Student.cpp file. The descriptors can be selected using their names (datatype: string). This is implemented in the function **descKeypoints.**

If user enters an invalid descriptor type, BRIEF detector will be executed by default.

```
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();
}
```

MP.5 Descriptor Matching:

FLANN matching and k-nearest neighbour selection was implemented. Both methods are selectable using the respective strings in the main function. If user enters an invalid matcher or selector type, Brute Force matcher and Nearest neighbour selector will be executed by default.

```
else if (matcherType.compare("MAT_FLANN") == 0)
{
    if ((descSource.type() != CV_32F) || (descRef.type() != CV_32F))
    {
        descSource.convertTo(descSource, CV_32F);
        descRef.convertTo(descRef, CV_32F);
    }

    matcher = cv::DescriptorMatcher::create(cv::DescriptorMatcher::FLANNBASED);
}
else
{
    std::cerr << "Enter a valid Matcher; Executing Default Matcher : Brute Force Matcher" << std::endl;
    int normType = (descriptorType == "DES_BINARY") ? cv::NORM_HAMMING : cv::NORM_L2;
    matcher = cv::BFMatcher::create(normType, crossCheck);
}
```

MP.6 Descriptor Distance Ratio:

Implemented 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 when using the KNN selector.

```
// 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);
    double minDescDistRatio = 0.8;

    for (auto & knn_match : knn_matches) {
        if (knn_match[0].distance < minDescDistRatio * knn_match[1].distance) {
            matches.push_back(knn_match[0]);
        }
    }
}
else
{
  std::cerr << "Enter a valid selector; Executing Default selector : NN selector" << std::endl;
  matcher->match(descSource, descRef, matches);
}
```

MP.7 Performance Evaluation 1: # keypoints on preceding vehicle

Number of Features in Vehicle Ahead using all Detectors												
Detector	Image 1	Image 2	Image 3	Image 4	Image 5	Image 6	Image 7	Image 8	Image 9	Image 10	Sum	Average
Harris	51	41	63	58	85	322	38	136	96	194	1084	108
Shi-Tomasi	125	118	123	120	120	113	114	123	111	112	1179	117
FAST	149	152	150	155	149	149	156	150	138	112	1460	146
BRISK	264	282	282	277	297	279	289	275	266	254	2765	276
ORB	92	102	106	113	109	125	130	129	127	128	1161	116
AKAZE	166	157	161	155	163	164	173	175	177	179	1670	167
SIFT	138	132	124	137	134	140	137	148	157	137	1384	138

MP.8 Performance Evaluation 2 : # of matched keypoints

Average Number of Macthes over 9 images						
Detector \ Descriptor	BRISK	BRIEF	ORB	FREAK	AKAZE	SIFT
Harris	31	34	36	29	NA	34
Shi-Tomasi	73	90	85	63	NA	103
FAST	86	98	95	74	NA	116
BRISK	144	149	103	121	NA	182
ORB	72	50	58	38	NA	84
AKAZE	123	120	102	108	130	141
SIFT	59	66	out of memory	56	NA	88

MP.9 Performance Evaluation 3 :

Detection Time:

Average Detector Time (in ms) over 10 images						
Detector \ Descriptor	BRISK	BRIEF	ORB	FREAK	AKAZE	SIFT
Harris	18.32	17.55	17.02	14.2	NA	17.03
Shi-Tomasi	18.73	18.02	17.55	14.11	NA	14.5
FAST	0.96	0.98	0.95	1.01	NA	0.91
BRISK	369.54	368.46	366.03	367.73	NA	368.2
ORB	9.74	8.13	8.51	8.09	NA	8.01
AKAZE	90.27	89.38	89.32	80.29	85.49	85.51
SIFT	114.01	135.33	out of memory	133.36	NA	108.79

Extraction Time:

Average Extractor Time (in ms) over 10 images						
Detector \ Descriptor	**BRISK**	**BRIEF**	**ORB**	**FREAK**	**AKAZE**	**SIFT**
Harris	26.32	1.51	1.05	41.42	NA	23.92
Shi-Tomasi	2.23	1.49	1.44	40.75	NA	17.54
FAST	2.18	1.26	1.27	42.81	NA	28.41
BRISK	3.42	1.34	4.86	42.55	NA	46.58
ORB	1.55	0.688	5.07	43.81	NA	51.56
AKAZE	2.42	1.01	3.69	42.89	77.1	26.65
SIFT	1.67	0.97	out of memory	43.88	NA	87.41

Best 3 Detector and Descriptor Combinations:

From the time taken and the number of matches obtained within that time frame, the following top 3 Detector and descriptor combinations shall be used.

1. FAST (Detector) + BRIEF (Descriptor)
2. FAST (Detector) + ORB (descriptor)
3. AKAZE (Detector) + BRIEF (Descriptor)

Notes:

1. All details in the table mentioned are available in the Performance.xls attached in the same folder.
2. AKAZE descriptor will only work if, AKAZE detector is used.