Chapter 1

Udacity Radar Engineering Midterm Report

1.1 MP.1 Data Buffer

Implements a ring buffer where new elements are added to tail and older are removed from head. The code checks if dataBuffer is equal to dataBufferSize. If so, then the first element will be removed by using erase method and the new element added to the tail (push back).

```
// I check if buffer is "full" if so, erase first image
        if (dataBuffer.size() == dataBufferSize)
        {
          dataBuffer.erase(std::begin(dataBuffer));
        }
        dataBuffer.push_back(frame);
```

1.2 MP.2 Keypoint Detection

With an **if** and **else** in the MidTermProjectCameraStudent.cpp the Shi-Tomasi, Harris, ORB, FAST, BRISK, AKAZE, and SIFT detectors are executed. For this the detectorType will be compared against the string "HARRIS" or "SHITOMASI". If the strings are equal, for example **detKeypointsHarris** function will be called. Otherwise, the **detKeypointsModern** function will be executed.

```
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);
}
```

1. The code below shows a part of the **detKeypointsModern** implementation.

```
    double t;
    cv::Ptr<cv::FeatureDetector> detector;

    if (detectorType.compare("FAST") == 0) {
        // TYPE_9_16, TYPE_7_12, TYPE_5_8
        cv::FastFeatureDetector::DetectorType type
                              cv::FastFeatureDetector::TYPE_9_16;
        detector = cv::FastFeatureDetector::create(30, true, type);

    } else if (detectorType.compare("BRISK") == 0) {
        detector = cv::BRISK::create();

    } else if (detectorType.compare("ORB") == 0) {
        detector = cv::ORB::create();

    } else if (detectorType.compare("AKAZE") == 0) {
        detector = cv::AKAZE::create();

    } else if (detectorType.compare("SIFT") == 0) {
        detector = cv::xfeatures2d::SIFT::create();
    }
```

2. Traditional Harris detector for keypoints detection is given by this function.

```
    int blockSize = 2;
    int apertureSize = 3;
    int minResponse = 100;
    double k = 0.04;

    cv::Mat dst, dst_norm, dst_norm_scaled;
    dst = cv::Mat::zeros(img.size(), CV_32FC1);

    double t = (double)cv::getTickCount();

    cv::cornerHarris(img, dst, blockSize, apertureSize, k,
                     cv::BORDER_DEFAULT);
    cv::normalize(dst, dst_norm, 0, 255, cv::NORM_MINMAX,
                  CV_32FC1, cv::Mat());
    cv::convertScaleAbs(dst_norm, dst_norm_scaled);

    // look for prominent corners and keypoints
    double maxOverlap = 0.0;
    for(size_t i = 0; i < dst_norm.rows; i++)
    {
        for(size_t j = 0; j < dst_norm.cols; j++)
        {
            int response = (int)dst_norm.at<float>(i,j);
            if(response > minResponse)
            {
                // only store points above a threshold
                cv::KeyPoint newKeypoint;
                newKeypoint.pt = cv::Point2f(j, i);
                newKeypoint.size = 2*apertureSize;
                newKeypoint.response = response;
                newKeypoint.class_id = 1;

                //ignore the rest of code
                }
             }
        }
```

1.3 MP.3 Keypoint Removal

With the code below only points within the defined range are used. 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 inPoints vector. In the end all points from the inPoints vector assigned to keypoints.

```
1  if (bFocusOnVehicle)
2  {
3     std::vector<cv::KeyPoint> inPoints;
4
5     for (auto point : keypoints)
6     {
7        if (vehicleRect.contains(cv::Point2f(point.pt)))
8        {
9           inPoints.push_back(point);
10       }
11    }
12
13   keypoints = inPoints;
14 }
```

1.4 MP.4 Keypoint Descriptors

Here is the same principle as in exercise 2. A parameter called **detectorType** makes the implemented descriptors BRIEF, ORB, FREAK, AKAZE and SIFT selectable.

```
void descKeypoints(vector<cv::KeyPoint> &keypoints,
                   cv::Mat &img, cv::Mat &descriptors,
                   string descriptorType)
{
    cv::Ptr<cv::DescriptorExtractor> extractor;

    if (descriptorType.compare("BRISK") == 0){
        int threshold = 30;
        int octaves = 3;
        float patternScale = 1.0f;

        extractor = cv::BRISK::create(threshold,
                                      octaves, patternScale);
    }
    else if(descriptorType.compare("SIFT") == 0)
    {
        extractor = cv::xfeatures2d::SiftDescriptorExtractor
                                    ::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("BRIEF") == 0)
    {
       extractor = cv::xfeatures2d::BriefDescriptorExtractor
                                    ::create();
    }
    extractor->compute(img, keypoints, descriptors);
}
```

1.5 MP.5 Descriptor Matching

```
if (matcherType.compare("MAT_BF") == 0)
{
  int normType = descriptorType.compare("DES_BINARY") == 0
                ? cv::NORM_HAMMING : cv::NORM_L2;
  matcher = cv::BFMatcher::create(normType, crossCheck);
}
else if (matcherType.compare("MAT_FLANN") == 0)
{
  if (descSource.type() != CV_32F)
  {
     descSource.convertTo(descSource, CV_32F);
     descRef.convertTo(descRef, CV_32F);
  }
    //... TODO : implement FLANN matching
matcher = cv::DescriptorMatcher::create
                      (cv::DescriptorMatcher::FLANNBASED);
}
```

1.6 MP.6 Descriptor Distance Ratio

Implemented k-nearest neighbor selection in the code listed below. The method is selectable using the respective strings in the main function. Use the KNN 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.

```
else if (selectorType.compare("SEL_KNN") == 0)
{
   // k nearest neighbors (k=2)
   vector<vector<cv::DMatch>> knn_matches;
   double t = (double)cv::getTickCount();
   matcher->knnMatch(descSource, descRef, knn_matches, 2);
   t = ((double)cv::getTickCount() - t)/cv::getTickFrequency();
   std::cout << "KNN with n = " << knn_matches.size()
   << " matches in " << 1000*t/1.0 << " ms" << std::endl;
   MatchingTime.push_back(1000*t/1.0);

  //Implement k-nearest-neighbor matching and 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]);
     }
   }
```

1.7 MP.7 Performance Evaluation 1

To 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.

See Attachment MP7 on page 7.

1.8 MP.8 Performance Evaluation 2

To count the number of matched keypoints for all 10 images using all possible combinations of detectors and descriptors. In the matching step, use the BF approach with the descriptor distance ratio set to 0.8.

All the numbers and time statistics for each combination are the average value on 10 sequence of images. See Attachment MP8 and MP9 on page 10.

1.9 MP.9 Performance Evaluation 3

Based on the final results, the top 3 detector/descriptor combinations for this project are:

1. FAST + BRIEF
2. FAST + ORB
3. FAST + BRISK

I choosed the dectecors and descriptors listed above. The reason is, that all of them have a good executen time and 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.

In case you want an even higher detection rate, then its better to choose the BRISK detector. The downside of BRISK is, you have an significantly slower execution time.

See Attachment MP8 and MP9 on page 10. The choosed detecors are filled green.

MP.7 Keypoints Counting

Detectors	Image 0	Image 1	Image 2	Image 3	Image 4	Image 5	Image 6	Image 7	Image 8	Image 9
SIFT	138	132	124	137	134	140	137	148	159	137
AKAZE	166	157	161	155	163	164	173	175	177	179
ORB	92	102	106	113	108	125	130	129	127	128
BRISK	264	282	282	277	297	279	289	272	266	254
FAST	149	152	150	155	149	149	156	150	138	143
Harris	17	14	18	21	26	43	18	31	26	34
Shi-Tomasi	125	118	123	120	120	113	114	123	111	112

MP.8 and MP.9 Matched Keypoints Counting and Time (Average)

Detector + Descriptor	Detected Keypoints (average)	Detection Time (average ms)	Extraction Time (average ms)	Matched Keypoints (average)	Matching Time (average ms)
FAST + BRISK	1787	0,931	2,22	100	0,43
FAST + BRIEF	1787	0,97	1,402	117	0,387
FAST + ORB	1787	0,977	2,70	115	0,408
FAST + SIFT	1787	0,99	34	117	3,35
FAST + AKAZE	1787	0,998	80	123	3,12
FAST + FREAK	1787	1,013	44,16	101	0,423
Shi-Tomasi + BRISK	1342	17,9	2,24	86	0,34
Shi-Tomasi + BRIEF	1342	17,88	1,25	110	0,276
Shi-Tomasi + ORB	1342	17,81	1,05	101	0,301
Shi-Tomasi + SIFT	1342	14,58	28	103	2,49
Shi-Tomasi + AKAZE	1342	15,8	34,25	108	2,3
Shi-Tomasi + FREAK	1342	15,96	41,84	86	0,316
HARRIS + BRISK	172	20,21	1,35	16	0,123
HARRIS + BRIEF	172	20,29	1,046	19	0,108
HARRIS + ORB	172	19,85	0,899	17	0,112
HARRIS + SIFT	172	20,39	22,16	18	0,56
HARRIS + AKAZE	172	18,5	35,21	19	0,45
HARRIS + FREAK	172	17,31	41,14	16	0,109
BRISK + BRISK	2711	42,87	3,37	170	1,114
BRISK + BRIEF	2711	40,755	1,224	186	0,987
BRISK + ORB	2711	41,74	4,81	160	1,007
BRISK + SIFT	2711	42,16	49,85	180	5,95
BRISK + AKAZE	2711	40	78,34	156	5,4
BRISK + FREAK	2711	42,533	45,001	161	0,944
ORB + BRISK	500	9,68	1,59	81	0,29
ORB + BRIEF	500	8,175	0,69	56	0,27
ORB + ORB	500	8,4	5,28	83	0,328
ORB + SIFT	500	8,49	55,86	82	2,166
ORB + AKAZE	500	9	77	57	2,1
ORB + FREAK	500	8,58	43,056	48	0,198
AKAZE + BRISK	1343	89,207	2,404	134	0,487
AKAZE + BRIEF	1343	90,833	1,28	132	0,48
AKAZE + ORB	1343	87,7	3,2	130	0,448
AKAZE + SIFT	1343	93,03	32,23	139	3,255
AKAZE + AKAZE	1343	85	85	139	3,2
AKAZE + FREAK	1343	82,81	42,28	130	0,486
SIFT + BRISK	1384	115,01	1,69	60	0,38
SIFT + BRIEF	1384	138,18	0,812	80	0,347
SIFT + SIFT	1384	112,57	89,96	86	2,68
SIFT + AKAZE	1384	119	75	40	2,4
SIFT + FREAK	1384	134,64	42,55	62	0,37