MP.1 DATA BUFFER

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.

```
DataFrame frame;
frame.cameraImg = imgGray;
dataBuffer.push_back(frame);

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

MP.2 Keypoint Detection

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

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

}
else if (detectorType.compare("HARRIS")==0)
{
    detKeypointsHarris(keypoints, imgGray,det_time, false);
}
else
{
    detKeypointsModern(keypoints, imgGray, detectorType,det_time);
    cout<< "SIZE OF Keypoints() "<< keypoints.size() <<endl;
```

```
void detKeypointsModern(std::vector<cv::KeyPoint> &keypoints, cv::Mat &img, std::string detectorType, std::vector<float>& det_time, bool bVis)
{
    cv::Ptr<cv::FeatureDetector> detector;

    if (detectorType.compare("FAST") == 0) {


        int threshold = 10;
        bool bNMS = true;

        detector = cv::FastFeatureDetector::create(threshold, bNMS, cv::FastFeatureDetector::TYPE_9_16);
    }
    else if (detectorType.compare("SIFT") == 0)
    {
        detector = cv::xfeatures2d::SIFT::create();
    }
    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();
    }

    double t = (double)cv::getTickCount();
    detector->detect(img, keypoints);
    t = ((double)cv::getTickCount() - t) / cv::getTickFrequency();
    cout << detectorType << " with n= " << keypoints.size() << " keypoints in " << 1000 * t / 1.0 << " ms" << endl;

    det_time.push_back(1000 * t / 1.0);
}
```

```
void detKeypointsHarris(std::vector<cv::KeyPoint> &keypoints, cv::Mat &img, std::vector<float>& det_time, bool bVis)
{
    int blockSize = 2;
    int apertureSize = 3;
    double  k = 0.04;
    int minResponse = 100;
    double maxOverlap = 0.0;


    double t = (double)cv::getTickCount();
    cv::Mat dst ;
    cv::Mat dst_norm;
    cv::Mat dst_norm_scaled;

    dst = cv::Mat::zeros(img.size(), CV_32FC1); // Harris의 output


    // - cornerHarris를 이용하여 grey된 이미지 변경(dst)
    // - dst 를 normalize (dst_norm)
    // - dst_norm을 convertScaleAbs를 이용하여 절대 가ㅅ 결과를 8비트로 변환(dst_norm_scaled)
    // - dst_norm를 순회하면서 thresh보다 높은곳에 circle표시

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


    for (size_t j = 0; j < dst_norm.rows; j++) {
     for (size_t i = 0; i < dst_norm.cols; i++) {
        int response = (int)dst_norm.at<float>(j, i);


        if (response < minResponse) continue;

        cv::KeyPoint newKeyPoint;
        newKeyPoint.pt = cv::Point2f(i, j);
        newKeyPoint.size = 2 * apertureSize;
        newKeyPoint.response = response;
```

MP.3 Keypoint Removal

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

```
// only keep keypoints on the preceding vehicle
bool bFocusOnVehicle = true;
cv::Rect vehicleRect(535, 180, 180, 150);
if (bFocusOnVehicle)
{
    vector <cv::KeyPoint> AFKeypoints;

    for ( auto kp:keypoints )
    {
        if ( vehicleRect.contains(kp.pt))
        {
            AFKeypoints.push_back(kp);
        }
    }
    keypoints = AFKeypoints;


    FINAL.push_back(keypoints.size());

    // cout<<"***************"<< FINAL[0]<<"***************" <<endl;

}
```

MP.4 Keypoint Descriptors

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.

```
DataFrame frame;
frame.cameraImg = imgGray;
dataBuffer.push_back(frame);

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

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.

```cpp
// Find best matches for keypoints in two camera images based on several matching methods
void matchDescriptors(std::vector<cv::KeyPoint> &kPtsSource, std::vector<cv::KeyPoint> &kPtsRef, cv::Mat &descSource, cv::Mat &descRef,
                      std::vector<cv::DMatch> &matches, std::string descriptorType, std::string matcherType, std::string selectorType)
{
    // configure matcher
    // bool crossCheck = false;

    bool crossCheck = selectorType.compare("SEL_NN") == 0 ? true : false;
    cv::Ptr<cv::DescriptorMatcher> matcher;

    if (matcherType.compare("MAT_BF") == 0)
    {
        int normType = descriptorType.compare("DES_HOG") == 0 ? cv::NORM_L2 : cv::NORM_HAMMING;

        matcher = cv::BFMatcher::create(normType, crossCheck);
    }
    else if (matcherType.compare("MAT_FLANN") == 0)
    {
        matcher = cv::FlannBasedMatcher::create();
    }

    // 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)

        int k =2;
        vector<vector<cv::DMatch>> knn_matches;
        matcher->knnMatch(descSource, descRef, knn_matches, k);

        double minDescDistRatio = 0.8;
        for (auto itr = knn_matches.begin(); itr != knn_matches.end(); ++itr)
        {

            if ((*itr)[0].distance < minDescDistRatio * (*itr)[1].distance)
            {
                matches.push_back((*itr)[0]);
            }
        }
    }
}
```

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.

```
int k =2;
vector<vector<cv::DMatch>> knn_matches;
matcher->knnMatch(descSource, descRef, knn_matches, k);

double minDescDistRatio = 0.8;
for (auto itr = knn_matches.begin(); itr != knn_matches.end(); ++itr)
{

    if ((*itr)[0].distance < minDescDistRatio * (*itr)[1].distance)
    {
        matches.push_back((*itr)[0]);
    }
```

MP.7 & MP.8 & MP.9

1. SHI-TOMASI

	BRISK	BRIEF	ORB	FREAK	AKAZE	SIFT
matched key-points	95 95 90 93 92 85 93 90 89	103 105 100 102 98 97 101 99 99	99 103 100 103 98 99 98 102 98	92 97 96 95 90 88 87 91 91		103 107 101 104 98 98 101 101 102
detection time	10.6821 10.0924 9.78302 8.7845 7.99449 8.41452 8.55278 7.54182 7.74824 8.07827	18.2285 11.2931 10.3625 7.51774 25.0318 23.8486 24.6581 21.7194 24.3391 11.5587	17.5954 10.0813 8.97448 8.0434 11.6452 25.8108 25.8594 26.1758 27.2749 23.0731	11.6108 9.02701 8.31513 7.5586 27.1041 28.4102 20.5345 15.4558 7.03131 7.05667		12.7424 8.85559 8.86673 11.8467 25.3764 24.4402 25.2516 25.9644 12.012 7.40269
descriptor extraction time	1.39963 1.29373 0.936192 0.912895 0.922102 0.910929 0.910644 0.974273 0.910208 0.872185	0.994112 0.757676 0.416628 0.339621 0.714834 0.642068 0.603589 0.865676 0.68159 0.318019	3.02048 2.35396 2.37913 2.30853 2.39472 4.283 4.53262 4.6774 4.95061 4.10442	25.9832 23.848 24.2393 25.3924 31.722 32.4208 33.6903 29.8191 25.4185 24.8295		13.6306 11.9756 8.78789 8.75545 14.003 13.3863 13.8038 15.0191 8.24022 10.2724

MP.7 & MP.8 & MP.9

2. HARRIS

	BRISK	BRIEF	ORB	FREAK	AKAZE	SIFT
matched key-points	12 12 13 16 21 15 15 24 21	13 13 15 18 24 16 16 25 24	13 13 14 19 24 16 15 24 23	12 13 15 18 21 17 17 20 19		13 13 15 18 25 16 16 24 24
detection time	19.2145 9.62101 9.37144 7.10335 7.63356 18.1796 7.90381 9.89842 8.25314 20.6107	15.5352 10.1821 9.98155 8.45035 24.2817 43.2815 16.5853 26.7299 26.3237 24.3451	19.3048 9.78396 10.1312 12.2798 28.4105 38.0538 23.7671 29.8933 26.9684 32.173	18.9387 10.1504 10.3287 7.71194 26.004 42.2781 13.0566 14.0516 28.2999 31.3852		11.0388 7.4876 10.2652 7.06771 12.3826 44.5758 29.573 26.8352 27.99 11.9291
descriptor extraction time	0.300487 0.640976 0.301255 0.317452 0.347823 0.478073 0.279476 0.382534 0.338513 0.39827	0.198253 0.52804 0.166419 0.188846 0.303835 0.305336 0.494504 0.338849 0.302859 0.295407	2.89687 2.2554 2.41533 2.26065 4.27941 2.70613 4.02896 3.9559 3.99619 3.39914	27.1095 23.5566 23.9758 25.5632 30.4317 24.9539 31.9334 24.2306 30.8734 27.937		10.7969 11.9398 8.4227 8.00002 8.13152 9.54703 13.7079 12.5719 13.4339 9.11099

MP.7 & MP.8 & MP.9

3. FAST

	BRISK	BRIEF	ORB	FREAK	AKAZE	SIFT
matched key-points	268 273 262 261 255 270 273 267 266	289 297 289 285 271 298 291 285 291	292 294 285 289 277 297 285 285 284	263 263 261 261 264 275 273 258 269		301 306 295 287 275 302 299 285 291
detection time	3.78332 1.73091 1.77292 1.54229 1.54534 1.63969 1.89139 1.72263 1.63558 1.65004	1.9782 1.66738 8.3374 8.54531 8.52379 8.46465 8.81915 8.08705 1.71373 1.58739	3.63531 3.14756 1.39246 1.46589 6.0051 9.25712 4.99542 8.93853 1.59483 1.63629	3.64264 1.68524 5.6393 8.78327 5.75531 8.49131 8.58624 8.57512 8.13769 8.33389		3.84082 1.64308 1.58797 5.98312 7.95037 8.11182 9.60947 9.40008 8.6345 8.9507
descriptor extraction time	3.36977 3.38615 2.72044 2.84644 2.61979 2.80897 2.76899 2.7321 2.66929 2.7258	1.50789 1.3392 3.22833 4.99178 3.30858 3.28098 7.4934 4.87065 1.85198 1.65478	5.07574 4.29285 2.78182 3.14626 3.3377 9.44676 5.73483 7.8585 2.94566 2.78837	35.129 29.8796 29.0008 42.2544 25.7458 45.8293 43.9512 38.684 39.4179 39.7232		22.9525 14.8838 14.5691 13.9789 25.7896 34.269 26.9108 30.6803 30.614 29.7138

MP.7 & MP.8 & MP.9

4. BRISK

	BRISK	BRIEF	ORB	FREAK	AKAZE	SIFT
matched key-points	170 185 172 176 178 195 178 178 168	174 172 162 167 172 184 179 177 163	180 185 163 173 179 185 178 175 170	159 177 161 170 159 176 168 172 152		166 180 161 173 166 181 177 169 163
detection time	26.9769 25.6105 25.4653 25.4807 25.2967 24.7613 24.8981 24.6583 25.2627 25.0863	27.085 26.687 25.6866 25.3595 25.4685 25.2929 25.8729 24.5286 25.1222 25.2768	27.4559 26.0033 25.4282 25.2991 25.1099 25.085 25.0659 25.0839 25.049 25.1023	26.8821 25.6408 25.3186 25.4963 25.1352 25.2278 25.681 24.5207 25.4567 25.6453		26.631 28.4423 25.5248 25.216 25.0795 25.2336 25.5641 24.6981 24.7232 25.4977
descriptor extraction time	2.28415 2.12282 2.03421 1.99576 2.10449 1.981 2.02923 1.98028 1.90629 1.86755	1.12158 0.622674 0.750149 1.05464 1.16868 1.08935 0.706817 1.11553 1.05944 1.14277	8.62932 9.17317 8.87123 11.8155 9.52542 9.33831 8.14494 9.1878 9.65051 8.38437	26.3785 27.1944 26.7113 27.4187 25.7231 27.3876 26.0617 25.6878 24.9405 24.7034		20.2954 16.314 19.7572 18.6416 22.2011 20.9529 22.0998 20.1599 21.3904 17.5891

MP.7 & MP.8 & MP.9

5. ORB

	BRISK	BRIEF	ORB	FREAK	AKAZE	SIFT
matched key-points	67 68 70 81 79 88 88 89 87	47 54 51 58 60 68 69 73 70	67 77 74 83 85 96 94 91 93	39 45 43 44 41 48 50 47 56		63 70 69 74 83 89 94 87 89
detection time	101.525 5.41463 6.3542 4.66241 5.15829 4.56582 4.53714 4.67978 4.67562 4.76389	73.1636 5.13919 6.19462 20.5667 21.0084 21.1931 18.0084 17.8773 21.7084 19.7462	79.3463 5.04949 4.91642 4.59162 9.40403 20.9574 22.0113 20.7631 18.8552 20.3638	102.769 5.15472 9.00859 21.4227 20.972 20.0819 19.2997 21.2073 6.04998 4.79584		94.8587 4.70977 5.28594 19.2904 19.7707 13.0974 18.3768 15.7509 17.5505 19.2529
descriptor extraction time	1.16531 0.830876 0.845455 0.882772 0.865263 0.959281 1.02878 1.01978 1.01585 1.00858	0.746437 0.307623 0.468495 0.883026 0.886836 0.994542 0.760616 0.754777 0.935294 0.751935	9.26492 8.53312 8.45772 8.17923 8.25423 16.2723 16.2672 17.791 21.6738 15.3951	26.105 25.5824 26.1896 32.5032 33.6477 34.3897 33.7423 34.7513 25.3416 24.5585		20.2487 18.025 19.7258 26.2096 26.3056 29.9069 30.5535 28.9017 29.7825 44.0827

MP.7 & MP.8 & MP.9

6. AKAZE

	BRISK	BRIEF	ORB	FREAK	AKAZE	SIFT
matched key-points	140 128 133 134 138 140 145 151 147	139 131 132 129 131 143 151 151 146	127 131 127 128 129 137 140 143 144	141 132 134 129 127 142 148 150 149	133 135 127 131 131 141 148 149 155	136 135 134 137 140 142 151 149 153
detection time	64.145 36.2447 38.6507 36.3632 37.8569 38.1313 37.8273 36.8136 35.7927 34.7716	40.8953 35.3858 37.007 58.8941 62.1707 45.8426 58.5678 55.443 59.0517 53.7511	48.15 39.5028 39.9545 45.9517 62.4178 59.0123 61.1379 59.7441 60.8198 60.9713	54.5869 37.3094 40.8815 36.3541 30.6602 57.8309 54.3363 55.8188 57.127 56.33	40.1077 37.5429 36.2641 55.3152 57.5464 52.0505 85.9756 80.645 39.9248 95.1619	50.5852 38.7279 39.4227 62.6152 58.7124 46.0549 67.8849 62.5632 69.6704 44.8777
descriptor extraction time	1.28999 1.15901 1.19133 1.12776 1.27461 1.20372 1.28794 1.30114 1.35541 1.29508	0.842341 0.480401 0.524565 0.482539 0.514334 0.517956 0.538713 0.54434 0.553765 0.532456	8.11981 5.82651 6.13628 5.72116 5.74624 5.78298 5.99779 7.22243 6.10314 6.11435	26.324 24.4356 25.8642 24.8472 24.3103 25.9548 25.8119 25.788 25.9943 25.7557	28.0014 30.1648 30.0954 28.9898 29.2133 38.354 29.9376 32.2485 32.1614 32.5526	12.4163 12.3786 13.2866 12.5077 13.5798 14.4934 12.1615 17.1283 12.1856 15.811

MP.7 & MP.8 & MP.9

7. SIFT

	BRISK	BRIEF	ORB	FREAK	AKAZE	SIFT
matched key-points	76 70 76 78 81 76 71 89 85	83 76 83 89 83 81 74 94 90		75 72 76 79 78 72 65 82 84		91 80 89 93 96 86 90 105 102
detection time	71.6791 66.0115 67.0647 65.0113 64.0486 65.0095 65.2091 65.7216 66.7073 70.3735	67.6294 66.1045 67.6361 88.3915 82.184 73.0779 87.1699 69.5045 87.364 65.1274		72.0897 65.1516 68.7092 89.9791 64.9196 83.1064 66.1 86.2377 69.9914 81.8641		69.9163 52.6823 54.0931 80.0748 52.5816 70.7912 62.2254 59.6904 77.6365 59.0408
descriptor extraction time	1.05526 0.99044 1.30637 1.03025 0.987503 1.01783 1.03848 1.0701 1.12753 1.01868	0.465601 0.684518 0.477326 0.710019 0.695426 0.698751 0.697954 0.734573 0.785639 0.697785		24.6027 24.2737 25.5582 24.291 24.3314 24.1768 24.1949 24.1997 24.3273 24.2593		56.6256 43.0921 42.7257 42.9287 44.3303 42.6888 45.6506 45.7623 45.6057 43.504

Conclusion

TOP 3 combinations

1. FAST + BRIEF (BEST RUN TIME)
2. ORB +ORB (ROBUST IN SCALE CHANGE)
3. SHI-TOMASI + BRISK (BEST MATCH)