Luis Angel Cabral Guzmán
Linkedin: www.linkedin.com/in/luis-angel-cabral-guzmán-07a41671

Camera Based 2D Feature Tracking Report

MP.1 Data Buffer Optimization: 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.

From line 177 to line 182 in “MidTermProject_Camera_Student.ccp” file is implemented the next code:

```
177            if(dataBuffer.size() == dataBufferSize)
178            {
179                dataBuffer.erase(begin(dataBuffer));
180            }
181
182            dataBuffer.push_back(frame);
```

At line 117 the “if” condition checks the dataBuffer size and if it is equal to the datafufferSize value (2) the front of the vector is removed at line 179 using erase method, after that at line 182 the next frame is pushed to the back.

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

From line 197 to 208 in “MidTermProject_Camera_Student.ccp” file the SHI-TOMASI, HARRIS, FAST, BRISK, ORB, AKAZE, and SIFT detector are executed, see the image bellow:

```
197            if (timeInformation[timeInformationIndex].detectorType.compare("SHITOMASI") == 0)
198            {
199                collectedData = detKeypointsShiTomasi(keypoints, imgGray, false);
200            }
201            else  if (timeInformation[timeInformationIndex].detectorType.compare("HARRIS") == 0)
202            {
203                collectedData = detKeypointsHarris(keypoints, imgGray, false);
204
205            } else
206            {
207                collectedData = detKeypointsModern(keypoints, imgGray, timeInformation[timeInformationIndex].detectorType, false);
208            }
```

At line 201 the “detecotrType” variable is compared with HARRIS string after that the “detKeypointsHarris” function is called, it “detecotrType” is not equal to HARRIS string at line 207 “detKeypointsModers” function is called, inside of this function other if-else conditions compare the “detecotrType” variable with FAST, BRISK, ORB, AKAZE, and SIFT string in order to execute the proper detector, “detKeypointsModers” function is in “matching2D_Student.cpp” file form line 263 to 326.

In “detKeypointsModers” method in the file “matching2D_Student.cpp” from line 269 to 296 the detector is selected according to “detectorType”, after that at line 310 the “detector->detect” method is executed with the grayscale image as input and in the “keypoints” vector the detected keypoints are stored, in the next image you can see the “detKeypointsModers” method.

```
CollectedData detKeypointsModern(std::vector<cv::KeyPoint> &keypoints, cv::Mat &img, std::string detectorType, bool bVis)
{
    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();
                    }
                    else
                    {
                        {/* code */}
                    }
                }
            }
        }
    }

    t = static_cast<double>(cv::getTickCount());
    detector->detect(img, keypoints);
    t = (static_cast<double>(cv::getTickCount()) - t) / cv::getTickFrequency();

    if (bVis)
    {
        const std::string windowName(detectorType + " detection results.");
        cv::Mat visImage{ img.clone() };
        cv::drawKeypoints(img, keypoints, visImage, cv::Scalar::all(-1), cv::DrawMatchesFlags::DRAW_RICH_KEYPOINTS);
        cv::namedWindow(windowName, 5);
        imshow(windowName, visImage);
        cv::waitKey(0);
    }

    collectedData.numKeyPoints = (int)keypoints.size();
    collectedData.elapsedTime = ((1000 * t) / 1.0);
    return collectedData ;
}
```

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

In “MidTermProject_Camera_Student.ccp” at line 219 a bounding box named “vehicleBox“ is created that is used in the “if condition” at line 226 witch extracted all keypoints founded within the bounding box defined in “vehicleBox”, all keypoint within the “vehicleBox” bounding box are stored in the “keypointsInVehicleBox” auxiliary vector if the “vehicleBox.contains” method returns a true value after checks if the point is within the boundary.When the “for loop” from line 224 to 230 finishes the "keypoints" vector is assigned with all keypoints stored in "keypointsInVehicleBox" vector. You can see the code in the next image:

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

            for (auto point : keypoints)
            {
                if (vehicleBox.contains(cv::Point2f(point.pt)))
                {
                    keypointsInVehicleBox.push_back(point);
                }
            }

            keypoints = keypointsInVehicleBox;

            timeInformation[timeInformationIndex].pointsLeftOnImage.at(imgIndex) = keypoints.size();
            std::cout << std::endl;
        }
```

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

In “matching2D_Student.cpp” file from line 89 to 136 a sequence of “if-else” conditions compare the “descriptorType” variable with BRIEF, ORB, FREAK, AKAZE, and SIFT strings, when the “descriptorType.compare” method return a true value the generic extractor variable is assigned with the appropriate descriptor, and the compute method is called in the line 140.

```
85  CollectedData descKeypoints(vector<cv::KeyPoint> &keypoints, cv::Mat &img, cv::Mat &descriptors, string descriptorType)
86  {
87      // select appropriate descriptor
88      cv::Ptr<cv::DescriptorExtractor> extractor;
89      if (descriptorType.compare("BRISK") == 0)
90      {
91          int threshold = 30;        // FAST/AGAST detection threshold score.
92          int octaves = 3;           // detection octaves (use 0 to do single scale)
93          float patternScale = 1.0f; // apply this scale to the pattern used for sampling the neighbourhood of a keypoint.
94
95          extractor = cv::BRISK::create(threshold, octaves, patternScale);
96      }
97      else
98      {
99          if (descriptorType.compare("ORB") == 0)
100         {
101             extractor = cv::ORB::create();
102         }
103         else
104         {
105             if (descriptorType.compare("FREAK") == 0)
106             {
107                 extractor = cv::xfeatures2d::FREAK::create();
108             }
109             else
110             {
111                 if (descriptorType.compare("AKAZE") == 0)
112                 {
113                     extractor = cv::AKAZE::create();
114                 }
115                 else
116                 {
117                     if (descriptorType.compare("SIFT") == 0)
118                     {
119                         extractor = cv::xfeatures2d::SIFT::create();
120                     }
121                     else
122                     {
123                         if (descriptorType.compare("BRIEF") == 0)
124                         {
125                             extractor = cv::xfeatures2d::BriefDescriptorExtractor::create();
126                         }
127                         else
128                         {
129                             /* code */
130                         }
131
132                     }
133                 }
134             }
135         }
136     }
```

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.

In “matching2D_Student.cpp” file from line 39 to 52 the FLANN matching is implemented, in the line 39, if the “matcherType.compare” method return a true value after compere with “MAT_FLANN” string, two if conditions are used to convert the descriptor matrices to CV_32F type if they are not CV_32F type to avoid an OpenCV, after that the “matcher” variable of “DescriptorMatcher” data type is assigned a pointer to a descriptor matcher constructed with a FLANNBASED type.

```
9   // Find best matches for keypoints in two camera images based on several matching methods
10  CollectedData matchDescriptors( std::vector<cv::KeyPoint> &kPtsSource,
11                                  std::vector<cv::KeyPoint> &kPtsRef,
12                                  cv::Mat &descSource, cv::Mat &descRef,
13                                  std::vector<cv::DMatch> &matches,
14                                  std::string descriptorType,
15                                  std::string matcherType,
16                                  std::string selectorType)
17  {
18      // configure matcher
19      bool crossCheck = false;
20      cv::Ptr<cv::DescriptorMatcher> matcher;
21      // configure matcher
22      double t;
23
24      if (matcherType.compare("MAT_BF") == 0)
25      {
26          int normType = cv::NORM_HAMMING;
27          matcher = cv::BFMatcher::create(normType, crossCheck);
28          std::cout << "BF matching cross-check = " << crossCheck << std::endl;
29
30          if (descRef.type() != CV_8U)
31          {
32              descRef.convertTo(descRef, CV_8U);
33          }
34          if (descSource.type() != CV_8U)
35          {
36              descSource.convertTo(descSource, CV_8U);
37          }
38      }
39      else if (matcherType.compare("MAT_FLANN") == 0)
40      {
41          if (descRef.type() != CV_32F)
42          {
43              descRef.convertTo(descRef, CV_32F);
44          }
45          if (descSource.type() != CV_32F)
46          {
47              descSource.convertTo(descSource, CV_32F);
48          }
49
50          matcher = cv::DescriptorMatcher::create(cv::DescriptorMatcher::FLANNBASED);
51          std::cout << "FLANN matching" << std::endl;
52      }
```

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.

In “matching2D_Student.cpp” file the K-Nearest-Neighbor selection is implemented, in line 68 a vector of “DMatch” type named “kNearestNeighborMatches” is used to store the matches from calling “matcher->knnMatch”, using a value of 2 for k, after that the descriptor distance ratio test is performed to 0.8 in each match in the “kNearestNeighborMatches” vector, for each point falling within the threshold distance is copied to the matches vector.

```
61      else if (selectorType.compare("SEL_KNN") == 0)
62      { // k nearest neighbors (k=2)
63          std::vector<std::vector<cv::DMatch>> kNearestNeighborMatches;
64          t = static_cast<double>(cv::getTickCount());
65
66          matcher->knnMatch(descSource, descRef, kNearestNeighborMatches, 2);
67
68          for (auto index{ std::begin(kNearestNeighborMatches) }; index != std::end(kNearestNeighborMatches); index = index+1)
69          {
70              if ((*index).at(0).distance < (0.8 * (*index).at(1).distance))
71              {
72                  matches.push_back((*index).at(0));
73              }
74          }
75
76          t = ((static_cast<double>(cv::getTickCount())) - t) / cv::getTickFrequency();
77      }
```

MP.7 Performance Evaluation 1: 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.

In line 234 in the “MidTermProject_Camera_Student.ccp” file the number of keypoints on the preceding vehicle are stored, the number of keypoints found on the vehicle is equal to the size of “keypointsInVehicleBox“ vector. At line 234 the number of keypoint in the vehicle are stored to write them in the CSV file. All detectors are implemented using the “for cycle” in line 145.

MP.8 Performance Evaluation 2: Count 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.

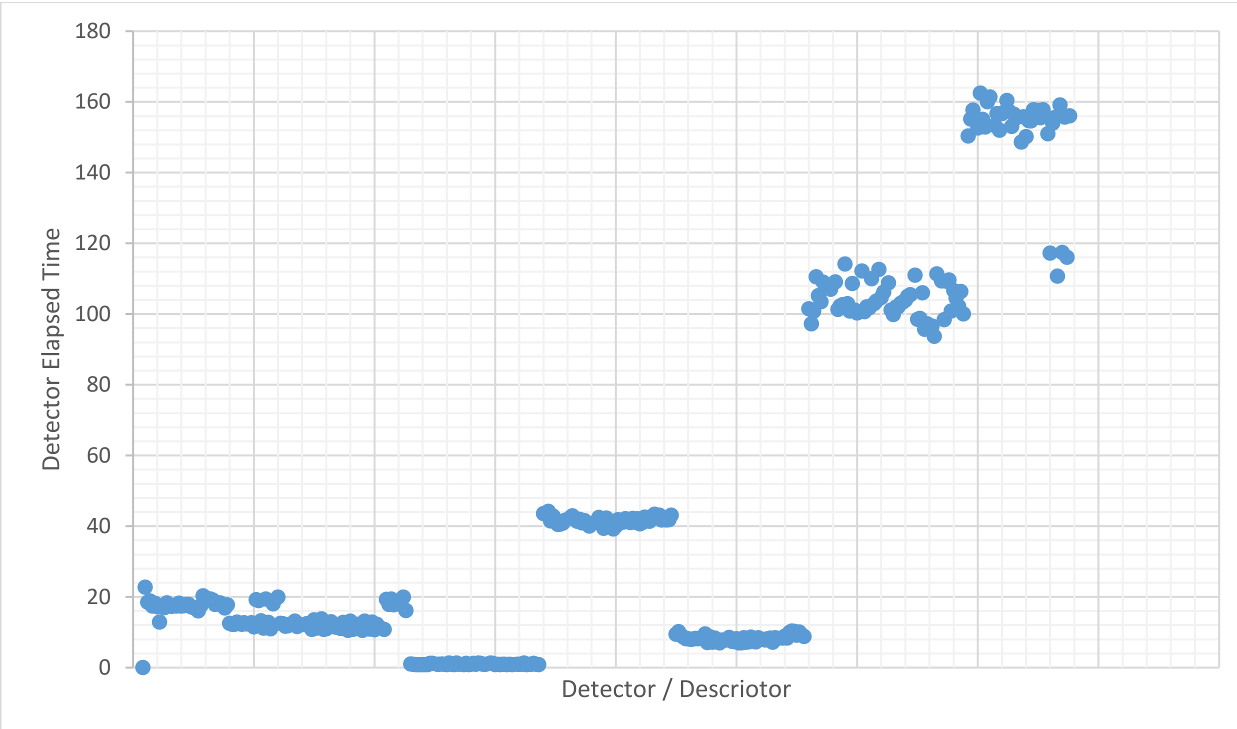
In the “matching2D_Student.cpp” file the size of “maches” vector at line 79 is the keypoints number using the distance ratio of 0.8, the “maches” size vector is stored to in the line 79 to write it in the CSV file. All detectors and descriptor are implemented using the “for cycle” in line 145 in “MidTermProject_Camera_Student.ccp” file.

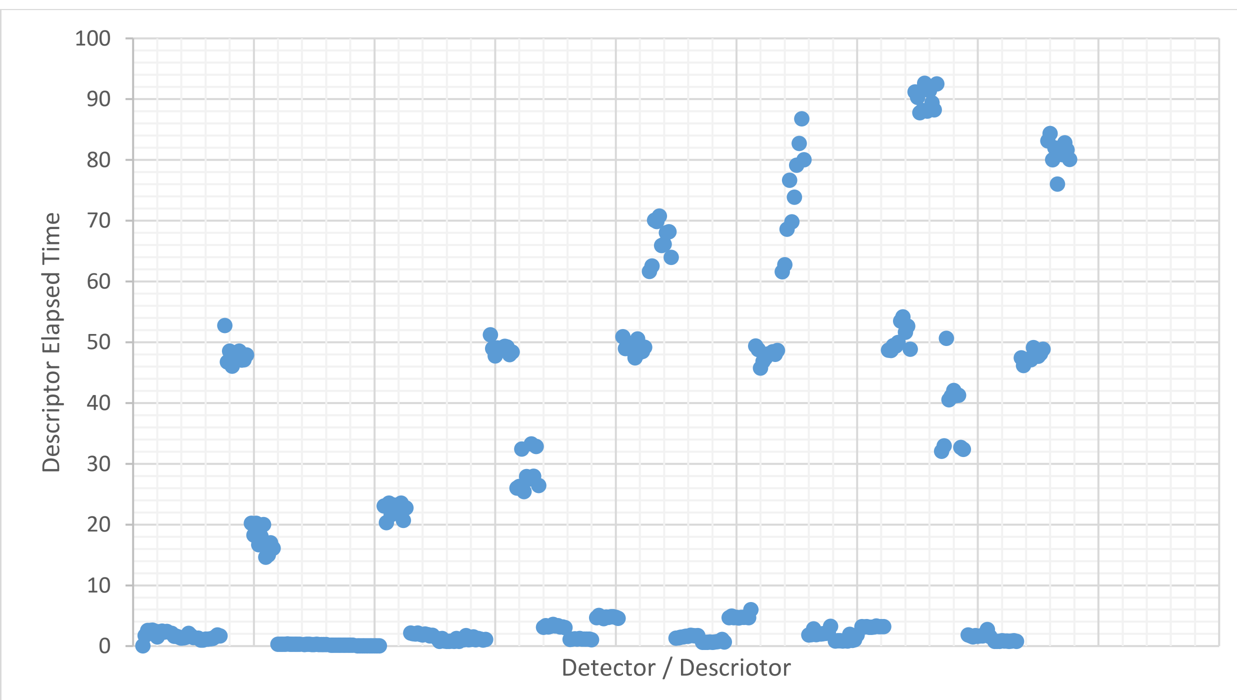
MP.9 Performance Evaluation 3: Log the time it takes for keypoint detection and descriptor extraction. The results must be entered into a spreadsheet and based on this data, the TOP3 detector / descriptor combinations must be recommended as the best choice for our purpose of detecting keypoints on vehicles.

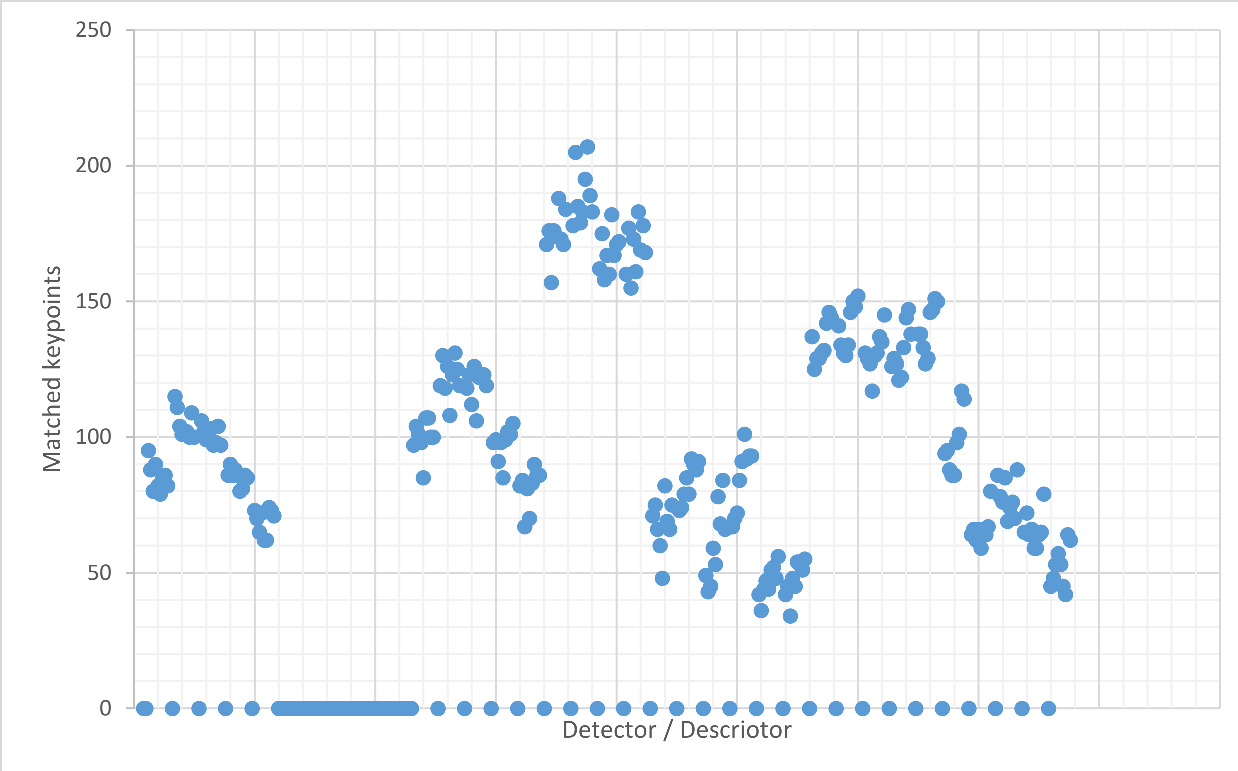
My top3 detector/descriptor are:

1. Fast detector / Brief descriptor
2. Fast detector / ORB descriptor
3. Fast detector / Brisk descriptor

These top 3 are in base of detector time and descriptor time that for all cases is not more than 3 secods, the high speed is important to real application, other important point is the matched keyPoints number, for those detector/ descriptor combination the matched points are around 100, I think they are enough, and the matcher time is close to 1 second., the CSV file are in the report folder.






250
200
150
100
50
0
Matched keypoints
Detector / Descriptor