Mid Term Project Writeup

MP.1 Data Buffer Optimization

Implement a vector for data Buffer 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.

- As we about to load new image, we need to check the size of the data buffer to see if it is at the limit. If so, we need to erase the first element in the data buffer (oldest image) and then we append the new image into the last place of data buffer.

MP.2 Keypoint Detection

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

- In the main script, we compare the input string “detectorType” with the name of each detector we defined to use them accordingly.

- Define the Harris detector following the guide from the lesson. For the other detector, we group them into **detKeypointsModern**. I refer to the guideline from OpenCV documentation to implement them one by one.

- Call detect() method for each detector to perform the search of keypoints in the current image.

MP.3 Keypoint Removal

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

- Use cv::Rect to define the ROI then loop through all keypoints and check if each keypoint is within the ROI to keep them. I used another vector “keypoints_sel” to keep track this process.

MP.4 Keypoint Descriptors

Implement descriptors BRIEF, ORB, FREAK, AKAZE and SIFT and make them selectable by setting a string accordingly.

- Follow the similar procedures as the step above, in the main script, we compare the input string “descriptorType” with all the options we have.

- Define each descriptor detector based on the guideline from OpenCV

- Call compute() method for each extractor to detect the descriptor in the current image

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 this step there are 2 types of matching distance need to implement. L2 norm is used for SIFT and Hamming distance is used for the binary descriptors

- Implement FLANN based method as introduced in the lesson.

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.

Implement the k nearest neighbor matching with k = 2. This method was introduced in the lesson.

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.

The table below shows the keypoints detected inside the ROI (target vehicle) by each detector for 10 frames provided. As we can see from this table: **BRISK**, **AKAZE** and **FAST** are the best candidates when it comes to the number of keypoints are detected.

	HARRIS	FAST	BRISK	ORB	AKAZE	SIFT
frame 1	17	149	264	92	166	138
frame 2	14	152	282	102	157	132
frame 3	18	150	282	106	161	124
frame 4	21	155	277	113	151	137
frame 5	26	149	297	109	163	134
frame 6	43	149	279	125	164	140
frame 7	18	156	289	130	173	137
frame 8	31	150	272	129	175	148
frame 9	26	138	266	127	177	159
frame 10	34	143	254	128	179	137

There are some detectors that have fixed neighborhood size:

- HARRIS: 6
- FAST: 7
- ORB: 31 – biggest size
- AKAZE: 4

BRISK also has big neighborhood size and big number of keypoints detected so they look busy.

SIFT also have pretty big neighborhood size but it has less number of keypoints so they look cleaner.

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.

The table below show the average matched descriptor with each combination of detector and descriptors:

As we can see from the table, the best performers are: **BRISK+ORB, AKAZE+ORB, FAST+ORB** when it comes to the number of matching descriptors detected.

Detector / Descriptor	HARRIS	FAST	BRISK	ORB	AKAZE	SIFT
BRIEF	16	36	29	24	36	23
ORB	16.11	96	102	59	102	out of memory
FREAK	14	28	28	24	31	19
AKAZE	N/A	N/A	N/A	N/A	44	N/A
SIFT	18	41	35	41	40	33

MP.9 Performance Evaluation 3

Log the time it takes for keypoints detection and descriptor extraction

The table below summaries the average time it takes for detecting keypoints on the second image + matching descriptors with the previous one (within the ROI). Based on it, the best performers are: **FAST+BRIEF, FAST+ORB, ORB+BRIEF**.

Detector / Descriptor	HARRIS	FAST	BRISK	ORB	AKAZE	SIFT
BRIEF	23.7	1.83	437	8.4	120	165
ORB	23	2	454	13.6	131	out of memory
FREAK	113	54.6	489	107	164	355
AKAZE	N/A	N/A	N/A	N/A	213	N/A
SIFT	52	27.9	480	72.7	153	224

Conclusion: Based on the 3 metrics, the best 3 configurations that I would pick are:

- **FAST + ORB**
- **FAST +BRIEF**
- **ORB+BRIEF**

I prefer the combination that can run fast and have a good amount of matching descriptor. The execution time is very important because we want to use it to calculate the TCC, a feature that need to run as fast as it can. Beside that, the combination that provide more matching can also good to make sure we calculate the changes between frames accurately.