Mid-Term Report

MP.1 Data Buffer Optimisation
Implemented RingBuffer component which contains limited length vector. This is achieved by pushing in new elements on one end and removing elements on the other end.
Also implemented RingBufferIterator with a limited number of public methods for more convenient work. All loaded from files images and computed keypoints, descriptors and matches
will be stored in ring buffer.

MP.2 Keypoints Detection
Implemented selectable by setting a string HARRIS, FAST, BRISK, ORB, AKAZE, and SIFT keypoints detectors. cv::FeatureDetector interface provides static method create() with different parameters for different cv::FeatureDetector implementations. It returns smart pointer on correct object. After called method detect() with image which returns detected keypoints vector. In case of Harris corners keypoints detector is used separate implemented method according to described algorithm.

MP.3 Keypoint Removal
Implemented the simple logic when we remove all keypoints outside of a pre-defined rectangle and only use the keypoints within the rectangle for further processing.

MP.4 Keypoint Descriptors
Implement selectable by setting a string descriptors BRIEF, ORB, FREAK, AKAZE and SIFT. cv::DescriptorExtractor interface provides static method create() with different parameters for different cv::DescriptorExtractor implementations. It returns smart pointer on correct object. Interface method compute() accepts keypoints vector and image and fills descriptor matrix.

MP.5 Descriptor Matching
Implemented Brute Force and FLANN matching. Brute-Force matcher is simple. It takes the descriptor of one feature in first set and is matched with all other features in second set using some distance calculation. And the closest one is returned. FLANN implements Fast Approximate Nearest Neighbor Search Library. The library builds a very efficient data structure (a KD-tree) to search for matching pairs and avoids the exhaustive search of the brute force approach
For both methods in possible to select nearest neighbours or k nearest neighbour selection.
Depending on descriptor type matching function selects distance metric, Hamming distance for binary descriptors and Euclidian distance for HOG descriptors.

MP.6 Descriptor Distance Ratio
For the K-Nearest-Neighbor matching 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.

MP.7 Performance Evaluation 1
Detected keypoints analysis

Detector type	Average count	Standart deviation
HARRIS	184,9	100,8022
FAST	4920,4	72,5027
BRISK	2711,6	48,9146
AKAZE	1342,9	16,1149
SIFT	1386,2	44,4111
ORB	500,0	0,0

FAST algorithm returns the largest keypoints vector. Keypoints count between neighbour frames has quite small variance. ORB detects always 500 keypoints because it is created with nFeatures option is equal 500. The less keypoints count and big variance gives Harris corners detection. Quite stable results shows AKAZE detector, keypoints count is almost the same between frames.

MP.8 Performance Evaluation 2

Detector/descriptor combination	Average count	Standart deviation
HARRIS/BRIEF	19,6667	5,3125
HARRIS/ORB	18,2222	4,5406
HARRIS/FREAK	16,0000	3,1972
HARRIS/SIFT	18,2222	4,4914
FAST/BRIEF	314,5556	17,1082
FAST/ORB	307,5556	11,6057
FAST/FREAK	248,1111	9,9827
FAST/SIFT	309,1111	11,7704
BRISK/BRIEF	189,3333	10,1325
BRISK/ORB	168,2222	7,2384
BRISK/FREAK	169,3333	8,8065
BRISK/SIFT	182,8889	9,1584
ORB/BRIEF	60,5556	13,6716
ORB/ORB	84,7778	11,4870
ORB/FREAK	46,6667	5,6372
ORB/SIFT	84,7778	9,5193
AKAZE/BRIEF	140,6667	8,1240
AKAZE/ORB	131,3333	7,1802
AKAZE/FREAK	131,8889	8,7996
AKAZE/SIFT	141,1111	8,1710
AKAZE/AKAZE	139,8889	8,5172
SIFT/BRIEF	78,0000	6,5149
SIFT/FREAK	65,8889	5,8962
SIFT/SIFT	88,8889	8,5172

The biggest number of matched keypoints gives FAST detector with SIFT or BRIEF descriptor. The Harris corner detects the minimal number of keypoints, so there are lower matches count. It is possible to estimate ratio between matches count and total keypoints count. It may show “quality” of detector.

MP.9 Performance Evaluation 3

Detector type	Average time, ms	Standart deviation
HARRIS	20,8500	6,2683
FAST	2,9163	0,2837
BRISK	44,4027	1,4848
AKAZE	86,5321	5,0026
SIFT	113,0779	11,5491
ORB	8,6413	0,5999

Detector/descriptor combination	Average time, ms	Standart deviation	Average time with detection, ms
HARRIS/BRIEF	0,8308	0,2419	21,6808
HARRIS/ORB	1,4094	0,0716	22,2594
HARRIS/FREAK	43,2024	6,1244	64,0524
HARRIS/SIFT	20,8785	2,7096	41,7285
FAST/BRIEF	2,5172	0,4566	23,3672
FAST/ORB	2,2909	0,2377	5,2072
FAST/FREAK	44,1578	1,6686	47,0741
FAST/SIFT	38,7642	3,0845	41,6805
BRISK/BRIEF	1,7793	0,3555	46,1820
BRISK/ORB	5,4078	0,4556	49,8105
BRISK/FREAK	42,6313	1,0624	87,0340
BRISK/SIFT	44,7218	4,1071	89,1245
ORB/BRIEF	1,0191	0,3859	9,6604
ORB/ORB	5,2829	0,3042	13,9242
ORB/FREAK	42,1874	1,8122	50,8287
ORB/SIFT	51,2669	8,1761	59,9082
AKAZE/BRIEF	1,6434	0,4747	88,1755
AKAZE/ORB	4,0058	0,5189	90,5379
AKAZE/FREAK	41,7347	1,7947	128,2668

Detector/descriptor combination	Average time, ms	Standart deviation	Average time with detection, ms
AKAZE/SIFT	26,4550	1,0517	112,9871
AKAZE/AKAZE	60,7377	1,5593	147,2698
SIFT/BRIEF	1,0376	0,1346	114,1155
SIFT/FREAK	42,7226	1,4777	155,8005
SIFT/SIFT	85,3310	8,2088	198,4089

If we estimate the time of detection of keypoints, then the most optimal are FAST, ORB and HARRIS, but the second and the third give significantly less detections count. In almost cases BRIEF descriptor gives better time comparing with another algorithms. According summary time (detection and descriptors computing) TOP3 looks like this FAST+ORB, ORB+BRIEF, ORB+ORB.
The reason may be the size of the descriptor (32 binary elements) and a smaller number of detected keypoints in comparison with other methods. For example by default SIFT descriptor contains from 128 floating point value, but it can be decreased if we will change initiation parameters. Also if we reduce keypoints number (filter somehow, for example using bounding boxes), descriptor computation time also will be reduced. But when we decrease descriptor size, it can affect on matching quality. For my opinion to select detector/descriptor pair developer should find compromise between computing time and results quality.