Udacity Radar Engineering Midterm Report

Brandon Marlowe

2019-09-14

MP.1 Data Buffer Optimization

In order to retain the ring buffer of size `dataBufferSize`, an `if` condition is used on line 140 of `MidTermProject_Camera_Student.cpp`. The `if` condition checks to see if the `dataBuffer` is equal to the `dataBufferSize`, and if so, the front of the vector is removed using the `erase` method, and the next frame is pushed to the back.

```
if (dataBuffer.size() == dataBufferSize) { dataBuffer.erase(std::begin(dataBuffer)); }

dataBuffer.push_back(frame);
```

MP.2 Keypoint Detection

Within an `if`, `else if`, and `else` conditions, the `Shi-Tomasi`, `Harris`, `ORB`, `FAST`, `BRISK`, `AKAZE`, and `SIFT` detectors are executed. On line 160 of `MidTermProject_Camera_Student.cpp`, the `detectorType` is compared against the string of `HARRIS`, and if the strings are equal, the `detKeypointsHarris` function is called. Otherwise, the `detKeypointsModern` function is executed, which contains a series of `if` conditions that compare the `detectorType` string against other string literals.

Within `detKeypointsModern`, a `cv::Ptr` of `FeatureDetector` type is assigned the proper detector based on the chosen detector. Once assigned, the `detector->detect` method is executed, which takes the grayscale image as input, and stores the detected keypoints in the `keypoints` vector. The results are displayed if the user has chosen to visualize them.

The number of keypoints detected and the elapsed time is returned in a struct, which is used to populate a vector, and in-turn fill out the final CSV file. This pattern is repeated with all function calls in the `matching2D_Student.cpp` file.

MP.3 Keypoint Removal

The keypoints found within the bounding box of the preceding vehicle are extracted within the `if` condition on line 177 of `MidTermProject_Camera_Student.cpp`. An auxilary vector named `retainedPoints` is used to store all the points found within the bounding box named `vehicleRect`. 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 `retainedPoints` vector. When the for-loop finishes, the `keypoints` vector is assigned the contents of the `retainedPoints` vector.

MP.4 Keypoint Descriptors

The `BRIEF, ORB, FREAK, AKAZE`, and `SIFT` descriptors were implemented in a manner similar to the detectors from `MP.2`. Beginning on line 57 of `matching2D_Student.cpp`, a series of `if` conditions are used to compare the string

input by the user. If the string matches one of the hard-coded descriptor types, the generic `extractor` variable is assigned the appropriate descriptor, and the `compute` method is called. Additionally, the elapsed time is measured during execution.

MP.5 Descriptor Matching

`FLANN` matching was implemented on line 21 of `matching2D_Student.cpp`. The `matcher cv:Ptr` of `cv::DescriptorMatcher` type is assigned a pointer to a descriptor matcher constructed with a `FLANNBASED` type. Additionally, two `if` conditions are used to determine if the descriptor matrices are not `CV_32F` type, and if so, they are converted in order to avoid an existing bug in OpenCV.

MP.6 Descriptor Distance Ratio

K-Nearest-Neighbor selection is implemented in `matching2D_Student.cpp` beginning on line 34. A 2D vector of `cv::DMatch` type is used to store the matches from calling `matcher->knnMatch`, using a value of 2 for `k`. Next, for each match in the `knnMatches` vector, the descriptor distance ratio test is performed. The distance threshold is set to 0.8, and each point falling within the threshold distance is copied to the `matches` vector.

MP.7 Performance Evaluation 1

The number of keypoints on the preceding vehicle are recorded within the `if` condition beginning on line 177. This task is accomplished as side-effect of task `MP.3`, where the number of keypoints found on the preceding vehicle is equal to the size of `retainedPoints` vector. The results are stored in the output CSV file.

MP.8 Performance Evaluation 2

The number of matched keypoints are determined in the `matchDescriptors` function in `matching2D_Student.cpp`. This is again accomplished as a side-effect of task `MP.6`, and the number of matched points is equal to the size of the `matches` vector. The results are stored in the output CSV file.

MP.9 Performance Evaluation 3

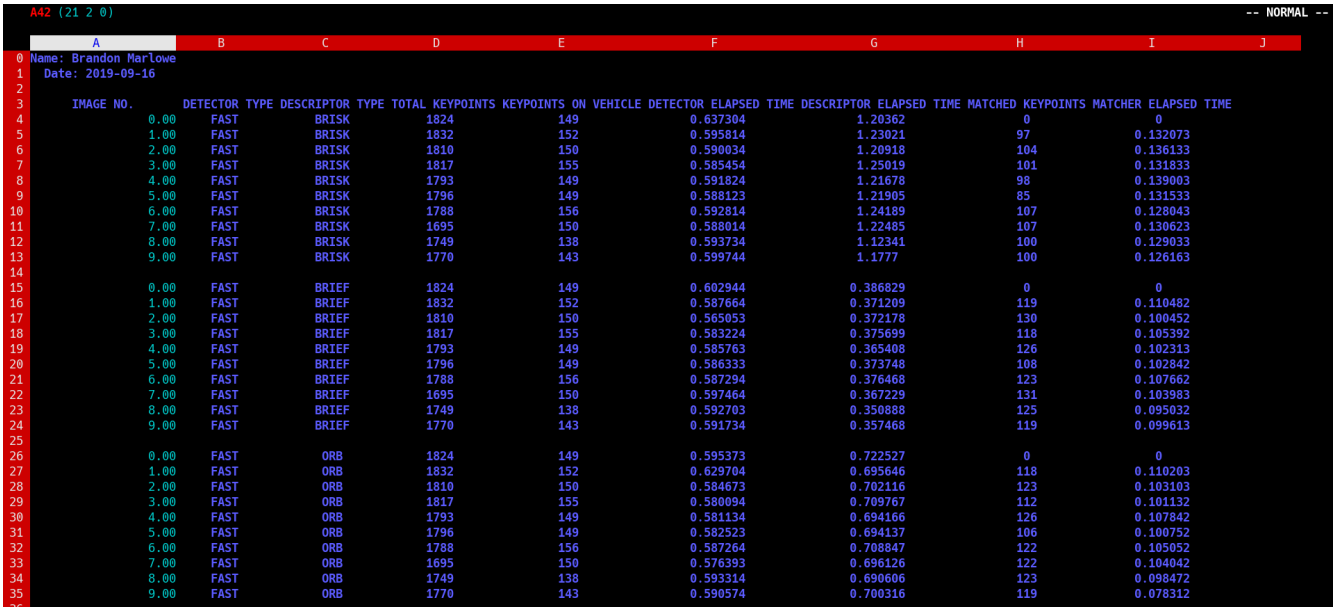
The final results can be found in the CSV file named `Brandon_Marlowe_Midterm_Project.csv` within the `report` directory. Based on the final results, the top 3 detector/descriptor combinations for this project are:

1. `Detector`: FAST, `Descriptor`: BRIEF
2. `Detector`: FAST, `Descriptor`: ORB
3. `Detector`: FAST, `Descriptor`: BRISK

The `FAST` detector in combination with the `BRIEF`, `ORB`, and `BRISK` descriptors executed in the shortest amount of time. 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. These three combinations retained much of the detail required to detect and track vehicles on the road. Reaction time for autonomous vehicles is extremely critical, thus having a performant dectector and descriptor combination is of utmost importance.

The BRISK detector was able to detect more initial points, and retain more points on the preceding vehicle, but with significantly slower execution time. In general, all other detector/descriptor combinations either executed more slowly and produced a similiar number of keypoints, or produced a larger number of keypoints, but with an even extremely slow execution time.

Note: The execution times shown in this screenshot are from running the program on my desktop computer. The timing data may vary when running in the student workspace.



A42 (21 2 0) -- NORMAL --

	A	B	C	D	E	F	G	H	I	J
0	Name: Brandon Marlowe									
1	Date: 2019-09-16									
2										
3	IMAGE NO.	DETECTOR TYPE	DESCRIPTOR TYPE	TOTAL KEYPOINTS	KEYPOINTS ON VEHICLE	DETECTOR ELAPSED TIME	DESCRIPTOR ELAPSED TIME	MATCHED KEYPOINTS	MATCHER ELAPSED TIME	
4	0.00	FAST	BRISK	1824	149	0.637304	1.20362	0	0	
5	1.00	FAST	BRISK	1832	152	0.595814	1.23021	97	0.132073	
6	2.00	FAST	BRISK	1810	150	0.590034	1.20918	104	0.136133	
7	3.00	FAST	BRISK	1817	155	0.585454	1.25019	101	0.131833	
8	4.00	FAST	BRISK	1793	149	0.591824	1.21678	98	0.139003	
9	5.00	FAST	BRISK	1796	149	0.588123	1.21905	85	0.131533	
10	6.00	FAST	BRISK	1788	156	0.592814	1.24189	107	0.128043	
11	7.00	FAST	BRISK	1695	150	0.588014	1.22485	107	0.130623	
12	8.00	FAST	BRISK	1749	138	0.593734	1.12341	100	0.129033	
13	9.00	FAST	BRISK	1770	143	0.599744	1.1777	100	0.126163	
14										
15	0.00	FAST	BRIEF	1824	149	0.602944	0.386829	0	0	
16	1.00	FAST	BRIEF	1832	152	0.587664	0.371209	119	0.110482	
17	2.00	FAST	BRIEF	1810	150	0.565053	0.372178	130	0.100452	
18	3.00	FAST	BRIEF	1817	155	0.583224	0.375699	118	0.105392	
19	4.00	FAST	BRIEF	1793	149	0.585763	0.365408	126	0.102313	
20	5.00	FAST	BRIEF	1796	149	0.586333	0.373748	108	0.102842	
21	6.00	FAST	BRIEF	1788	156	0.587294	0.376468	123	0.107662	
22	7.00	FAST	BRIEF	1695	150	0.597464	0.367229	131	0.103983	
23	8.00	FAST	BRIEF	1749	138	0.592703	0.350888	125	0.095032	
24	9.00	FAST	BRIEF	1770	143	0.591734	0.357468	119	0.099613	
25										
26	0.00	FAST	ORB	1824	149	0.595373	0.722527	0	0	
27	1.00	FAST	ORB	1832	152	0.629704	0.695646	118	0.110203	
28	2.00	FAST	ORB	1810	150	0.584673	0.702116	123	0.103103	
29	3.00	FAST	ORB	1817	155	0.580094	0.709767	112	0.101132	
30	4.00	FAST	ORB	1793	149	0.581134	0.694166	126	0.107842	
31	5.00	FAST	ORB	1796	149	0.582523	0.694137	106	0.100752	
32	6.00	FAST	ORB	1788	156	0.587264	0.708847	122	0.105052	
33	7.00	FAST	ORB	1695	150	0.576393	0.696126	122	0.104042	
34	8.00	FAST	ORB	1749	138	0.593314	0.690606	123	0.098472	
35	9.00	FAST	ORB	1770	143	0.590574	0.700316	119	0.078312	

Figure 1: