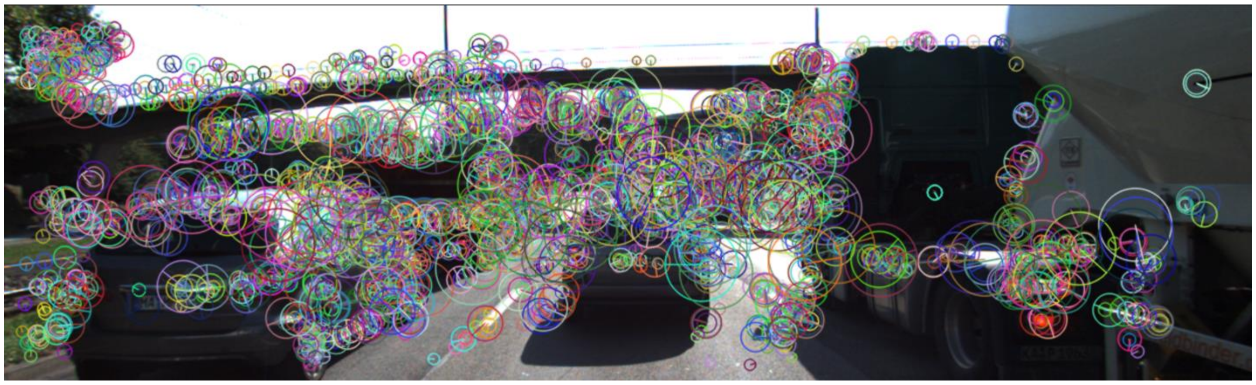
Sensor Fusion – 2D feature tracking - ReadMe

By Gaurav Borgaonkar

(25 Nov 2019)



The aim of this project is to implement and test different types of keypoint detector and keypoint descriptor algorithms and benchmark their performance in order to build a camera based collision detection system in the future.

The steps followed are as follows –

1. To begin with, we will be loading images, setting up data structures and putting everything into a ring buffer to optimize the memory load.
2. Then, several keypoint detectors such as HARRIS, FAST, BRISK and SIFT and compared with respect to number of keypoints and speed.
3. In the next part, we will then focus on descriptor extraction and matching using brute force approach and the FLANN approach.
4. In the last part, once the code framework is complete, we will test the algorithms in different combinations and compare them with respect to some performance measures.

Dependencies for Running Locally:

- cmake >= 2.8
  - All OSes: [click here for installation instructions] (https://cmake.org/install/)
- make >= 4.1 (Linux, Mac), 3.81 (Windows)
  - Linux: make is installed by default on most Linux distros
  - Mac: [install X-code command line tools to get make] (https://developer.apple.com/xcode/features/)
  - Windows: [Click here for installation instructions] (http://gnuwin32.sourceforge.net/packages/make.htm)
- OpenCV >= 4.1
  - This must be compiled from source using the `-D OPENCV_ENABLE_NONFREE=ON` cmake flag for testing the SIFT and SURF detectors.
  - The OpenCV 4.1.0 source code can be found [here] (https://github.com/opencv/opencv/tree/4.1.0)
- gcc/g++ >= 5.4
  - Linux: gcc / g++ is installed by default on most Linux distros
  - Mac: same deal as make - [install X-code command line tools] (https://developer.apple.com/xcode/features/)
  - Windows: recommend using [MinGW](http://www.mingw.org/)

Basic Build Instructions

1. Clone this repo.
2. Make a build directory in the top level directory: `mkdir build && cd build`
3. Compile: `cmake .. && make`
4. Run it: `./2D_feature_tracking`.

Project Rubric Points:

Task No	Task	Implementation Description
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.	Implemented a basic "if" loop that checks if the size of the Data Buffer vector with 2. If the size is more than 2, it will remove the element at the beginning of the vector. (i.e. The oldest image loaded is removed from the data buffer.)
2	Keypoint Detection - Implement detectors HARRIS, FAST, BRISK, ORB, AKAZE, and SIFT and make them selectable by setting a string accordingly.	Created a feature detector object. When the input string matches, that detector type is assigned to the object and executed. Wrote two functions, one for Harris Corner Detection and another for modern detectors FAST, BRISK, ORB, AKAZE and SIFT
3	Keypoints removal - Remove all keypoints outside of a pre-defined rectangle and only use the keypoints within the rectangle for further processing.	I created a new keypoint vector for filtered keypoints. If the keypoint is located in the specified (mostly center) rectangular space in the image, we add it to filtered keypoints vector.
4	Keypoint Descriptors - Implement descriptors BRIEF, ORB, FREAK, AKAZE and SIFT and make them selectable by setting a string accordingly.	Created a descriptor extractor object. When the input string matches, that descriptor type is assigned to the object and executed. Wrote cases for BRISK, BRIEF, ORB, FREAK, AKAZE and SIFT descriptors
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.	Created string variables to select descriptor matcher and implemented FLANN matching in matching2D_student.cpp file. Debugged the file for opencv bug by converting binary descriptors into floating point vectors
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.	Used k = 2 and a threshold value of 0.8 between True Positives and False Positives for descriptor matching

7	Your seventh task is to 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.	I created a variable **total_keypoints** that is updated over every instance of the for loop execution and gives us the total number of keypoints at the end of execution of the program.
8	Your eighth task is to count the number of matched keypoints for all 10 images using all possible combinations of detectors and descriptors. In the matching step, use the BF approach with the descriptor distance ratio set to 0.8.	I created a variable **total_matched_keypoints** that adds the number of keypoints matched in every single execution of the for loop (for every pair of images) and at the end of execution of the program, gives us total.
9	Your ninth task is to log the time it takes for keypoint detection and descriptor extraction. The results must be entered into a spreadsheet and based on this information you will then suggest the TOP3 detector / descriptor combinations as the best choice for our purpose of detecting keypoints on vehicles. Finally, in a short text, please justify your recommendation based on your observations and on the data you collected.	Similar to counting the number of keypoints, I created a time variable **total_time** that counts the time required for each pair of images for both detection and extraction and adds them together. This time variable adds times for all pairs of images together. After adding the times, we divide the total number of keypoints by total time required and calculate the efficiency of the detector/descriptor combination.

TASK MP. 7: To count the total number of keypoints on the preceding vehicle for every detector

Detector Type	Keypoints detected for the preceding vehicle
SHITOMASI	1179
HARRIS	176
FAST	4094
BRISK	2762
ORB	1161
AKAZE	1670
SIFT	1386

Performance Evaluation:

Fastest three detector-descriptor combinations have been highlighted in green.

Sr No	Detector + Descriptor	Total KeyPoints	Total Matches	Total Time (ms)	Ratio (matches/ms)
1	SHITOMASI + BRISK	13423	4912	338.80	14.50
2	SHITOMASI + BRIEF	13423	6625	219.72	30.15
3	SHITOMASI + ORB	13423	5673	220.91	25.17
4	SHITOMASI + FREAK	13423	4084	622.21	6.564
5	SHITOMASI + AKAZE	13423	NaN	NaN	NaN
6	SHITOMASI + SIFT	13423	9484	474.33	19.99
7	HARRIS + BRISK	728	410	202.91	2.02
8	HARRIS + BRIEF	728	477	180.05	2.65
9	HARRIS + ORB	728	427	193.54	2.21
10	HARRIS + FREAK	728	290	543.42	0.53
11	HARRIS + AKAZE	728	NaN	NaN	NaN
12	HARRIS + SIFT	728	554	384.82	1.439
13	FAST + BRISK	49204	14267	503.38	28.34
14	FAST + BRIEF	49204	19554	195.83	99.85
15	FAST + ORB	49204	16896	107.25	157.54
16	FAST + FREAK	49204	11717	799.77	14.65
17	FAST + AKAZE	49204	NaN	NaN	NaN
18	FAST + SIFT	49204	30082	2134.49	14.09
19	BRISK + BRISK	27116	9563	4139.32	2.31
20	BRISK + BRIEF	27116	11426	3924.53	2.91
21	BRISK + ORB	27116	7039	3879.17	1.81

22	BRISK + FREAK	27116	7780	4472.89	1.74
23	BRISK + AKAZE	27116	NaN	NaN	NaN
24	BRISK + SIFT	27116	15279	6428.55	2.38
25	ORB + BRISK	5000	2427	141.03	17.21
26	ORB + BRIEF	5000	1822	109.10	16.70
27	ORB + ORB	5000	1996	134.08	14.89
28	ORB + FREAK	5000	920	522.93	1.76
29	ORB + AKAZE	5000	NaN	NaN	NaN
30	ORB + SIFT	5000	3360	1520.63	2.21
31	AKAZE + BRISK	13429	6869	1007.71	6.81
32	AKAZE + BRIEF	13429	7008	915.58	7.65
33	AKAZE + ORB	13429	5111	940.72	5.43
34	AKAZE + FREAK	13429	5319	1313.15	4.05
35	AKAZE + AKAZE	13429	8036	1654.58	4.86
36	AKAZE + SIFT	13429	9489	1478.94	6.42
37	SIFT + BRISK	13860	4285	1391.25	3.08
38	SIFT + BRIEF	13860	4553	1357.28	3.36
39	SIFT + ORB	13860	Out of Memory Error	Out of Memory Error	Out of Memory Error
40	SIFT + FREAK	13860	3334	2334.01	1.43
41	SIFT + AKAZE	13860	NaN	NaN	NaN
42	SIFT + SIFT	13860	7020	2298.95	3.05

Created a variable that adds the number of all keypoints after every iteration. I also created a time variable that adds together all the times for detector descriptor executions for every image pair sequence.

TASK MP. 7

For this, I created a variable **total_preceding_vehicle_keypoints** which updates itself for every loop execution of the image pair sequence. At the end, it prints the value of total number of preceding vehicle keypoints for the particular detector on screen.

TASK MP. 8

We count the number of matched keypoints for every pair of image sequence and store it in a variable **total_matches.** This variable is updated for every pair of images and at the end, it prints a value of total number of matched keypoints.

TASK MP. 9

Top Three Detector/Descriptor pairs:

The 3 best performing detector/descriptor pairs are selected based on their performance. Performance is evaluated by no. of matches on the preceding vehicle divided by time required.

Note: - For object tracking, although we are concerned about the keypoints only on the preceding vehicle; I have calculated the efficiency of these algorithms by using keypoints from all over the image.

Sr No	Detector + Descriptor	Total KeyPoints	Total Matches	Total Time (ms)	Ratio (matches/time)
1	FAST + ORB	49204	16896	107.25	157.54
2	FAST+ BRIEF	49204	19554	195.83	99.85
3	SHITOMASI + ORB	13423	6625	219.72	30.15