Project Report

Author:Numan Abdullah

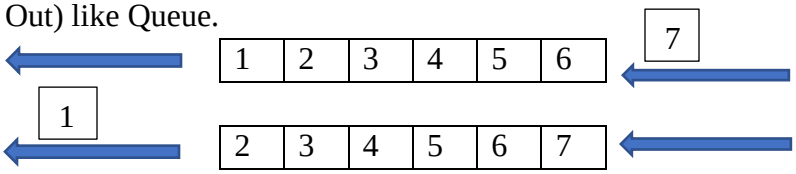
MP.1 Data Buffer Optimization

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.

For this task, vector of DataFrame is implemented. It is worked as buffer for storing the images in the memory to avoid performances deficiency. DataFrame is data structure that is composed of following parameter:

- Image
- 2D keypoints in the Image
- keypoint descriptors
- Matches between two Images(Consecutive)

Next, buffer size is set to 2 mean two images can be saved in the storage. This work like as FIFO(First In First Out) like Queue.



MP.2 Keypoint Detection

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

In this task, HARRIS, FAST, BRISK, ORB, AKAZE, and SIFT strings are compared and on the bases of specified term the respective detector configured in the matching_2Dstudent file's function that is called as detKeypointsModern(). Moreover, the Harris Corner based detector is programmed separately detKeypointsHarris().

Secondly, cv::Ptr of FeatureDetector type will assigne the proper detector based on the chosen detector string. After that, detector->detect is executed, to detect the keypoints and stored for further processing.

MP.3 Keypoint Removal

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

To consider only the keypoints in the specific Region of Interest(ROI) a cv::Rect functionality is used to make a bound to consider only those points that are in that particular area. For this task, a built in function as provided by OpenCV, Rect class method named as vehicleRect.contain() is used to check either that Rect object has the keypoint or not. If the particular keypoint is resided in that, the respective point is pushed to the newFiltered Vector of keypoint.

MP.4 Keypoint Descriptors

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

After getting keypoints, next is to use descriptors to describes the image patch around a keypoint to tackle scale change ,rotation, Intensity change and affine transformation. In the code, same technique is employed as in section MP.2. to implement BRIEF, ORB, FREAK, AKAZE, and SIFT descriptors. As in matching2D_Student.cpp, a combination of if conditions are to compare the string with specified string. Next, elapsed time and size of the resultant output is shown for analysis and observation.

.

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.

If there are N keypoints and their associated descriptors in one image and M keypoints in another image. Brute Force Matching or Nearest Neighbor Matching can be used to compare all features with each other, i.e. perform N x M comparisons. FLANN matching is issued by the SIFT author and this technique is using the most efficient data structure named as KD Tree that we implemented in LiDAR Sensor. In addition to it, the matcher cv:Ptr of cv::DescriptorMatcher type is assigned a pointer to a descriptor matcher constructed with a FLANNBASED type. Moreover, descriptor matrices are not of type CV_32F then those need to be changed to CV_32F.

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.

This method is used to lower the number of false positives. In it, for each keypoint in the source image, the two best matches are located in the reference image and the ratio between the descriptor distances is computed K-Nearest-Neighbor selection is implemented in matching2D_Student.cpp. 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

Count the number of Key-Points 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.

Image		0	1	2	3	4	5	6	7	8	9
Key-Points (preceding vehicle)											
SHITOMASI	BS=2	230	222	226	219	241	236	227	229	229	239
	BS=4	125	118	123	120	120	130	114	123	111	112
	BS=6	80	83	82	81	75	74	76	78	74	79
	BS=8	59	62	60	63	56	55	52	57	57	55
HARRIS	BS=2	27	26	29	31	55	96	22	66	60	81
	BS=4	43	44	55	50	69	57	53	110	73	65
	BS=6	87	21	43	23	76	55	22	67	45	56
FAST		419	427	404	423	386	414	418	406	396	401
BRISK		264	282	282	277	297	279	289	272	266	254
ORB		100	102	106	113	109	125	130	129	127	128
AKAZE		166	157	161	155	163	164	173	175	177	179
SIFT		132	124	124	137	134	140	137	148	159	137

With SHITOMASI Detector, the keypoint population is decreases with Block Size(Neigborhood). Moreover, Harris detector is unstable as with Block size there is variation in the resultant keypoints across image 0 to 9. FAST, BRISK, ORB, AKAZE, SIFT results is shown.

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.

As the trend can be interrupted in the attached CSV. For this all combination detector and descriptor is implemented iteratively, using the function defined in the matching2D_student.cpp. That includes the descKeypoints() for detection of descriptor and for detectors detKeypointsShiTomasi(), detKeypointsHarris() and detKeypointsModern() is employed. In the end next matchDescriptors() is employed with the given parameter that are using Nearest Neighbour Algorithm with distance ratio set to 0.8 to avoid false positive while selecting the keypoints for mathing.

MP.7 Performance Evaluation 1

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.

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

- ➔ Detector: FAST, Descriptor: BRIEF
- ➔ Detector: FAST, Descriptor: ORB
- ➔ Detector: FAST, Descriptor: BRISK

The FAST detector in with either one of the following:

- ➔ BRIEF
- ➔ ORB
- ➔ BRISK

With these combinations give good results and executed in the shortest amount of time. Moreover, they were able to maintain a good portion of points on the preceding vehicle and match a good portion of points between successive images. As Accuracy is critical in autonomous system and to make good prediction need large number of keypoints that resided in all the images. The BRISK detector was able to detect more initial points, and retain more points on the preceding vehicle, but with significantly slower execution time So there is tradeoff between execution time and performance.