Designing an Augmented Reality System Using ORB (Oriented FAST and Rotated BRIEF)

Submitted in partial fulfillment of the requirements of
BITS G540 Research Practice

By

Umang Dhiman
ID No. 2016H112151P

Under the supervision of
Dr. Mukesh Kumar Rohil
Associate Professor
Department of Computer Science and Information Systems
BITS Pilani, Pilani Campus

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI, PILANI CAMPUS

2 May, 2017

CERTIFICATE

This is to certify that the report entitled, Designing an Augmented Reality System Using ORB(Oriented FAST and Rotated BRIEF) and submitted by Umang Dhiman ID No. 2016H112151P in partial fulfillment of the requirement of BITS G540 Research Practice embodies the work done by him/her under my supervision.

Signature of the Supervisor

Date:

Name

Designation

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

ORB – Oriented FAST and rotated BRIEF

FAST - Features from Accelerated Segment Test

BRIEF - Binary Robust Independent Elementary Features

SIFT – Scale Invariant Feature Transform

SURF – Speedup Robust Features

LBP – Local Binary Mode

oFAST - FAST Keypoint Orientation

KNN- K Nearest Neighbors

rBRIEF- Rotation Aware BRIEF

FLANN - Fast Library for Approximate Nearest Neighbors

RANSAC – RANdom Sampling Consensus Algorithm

ABSTRACT

Oriented FAST and rotated BRIEF (ORB) is a binary descriptor useful for tasks like keypoints detection in an image, features detection and matching images. It is a technique which is resistant to noise and is invariant to rotation.

It can be used for Augmented Reality purposes. Augmented Reality makes a environment which seems real but rather it uses only virtual objects using the actual environment. It mixes virtual objects with the view of real world. To get the original appearance of the scene from distorted image, ORB can be used for creating augmented reality system.

We have implemented ORB for detecting keypoints and features in any image, and to match images which have been distorted in some way.

KEYWORDS: ORB, OPENCV, FAST, BRIEF, Brute Force Matching, Flann Based Matching, Ratio Based Matching, RANSAC, Homography Matrix

1. INTRODUCTION

In computer vision, keypoints detection, features description and matching are essential components. There are various costly and time consuming mechanisms for detecting and matching images. Previously there was SIFT detector and descriptor which has lots of applications now also but its computational cost is very high. Similarly there was SURF which was also computationally complex. So an alternative to fast binary descriptor based on BRIEF is ORB (Oriented FAST and Rotated BRIEF).

1.1. LITERATURE SURVEY

Paper[1] presents a technique to detect keypoints. These keypoints can then be easily matched due to the binary nature of the descriptor

Paper[2] presents a method for image matching which uses and enhances LBP(Local Binary Mode) point matching algorithm

Paper[5] ORB is mainly based on modified FAST and BRIEF :

1) FAST is a method for finding keypoints in an image. Keypoints are basically those interesting or distinguishing points in an image which define it. Even if the image is distorted in some form, we can still find the same keypoints in the distorted image.
2) BRIEF is a method to detect features. It uses binary tests between pixels in a smoothed image patch.

The FAST and BRIEF techniques have been modified as oFAST i.e. FAST Keypoint Orientation and rBRIEF i.e. Rotation Aware BRIEF due to various limitations.

1) oFAST - As FAST doesnot have orientation operator, and has large reponses along the edges. So due to that, a Harris corner measure is used along with it to order the keypoints and for corner orientation, intensity centroid is used
2) rBRIEF – As BRIEF performs poorly in the case of rotation. rBRIEF handles that case as well.

Keypoints and features of each image are the computed using oFAST and rBRIEF.

For keypoints and descriptors detection, the process can be formulated as a flow chart as shown :

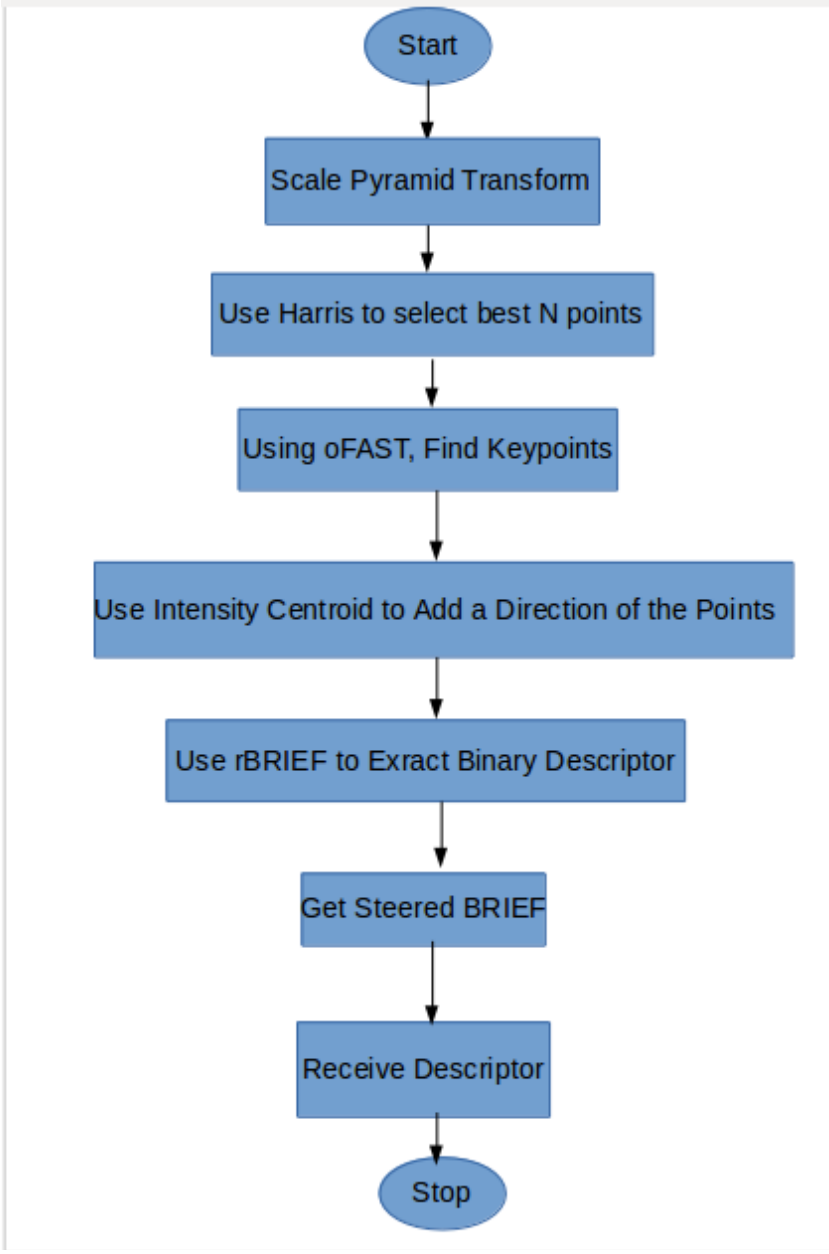


Fig. 1 Flowchart of ORB

After detection of keypoints and features, images were being matched using brute-force technique.

2. CONTRIBUTION

According to the paper [5], there has been a significant improvement in the time taken to compare images using ORB as compared to SIFT and SURF. They have used Brute force approach to find the number of matches between images. I have used various other approaches also like Distance based matching which uses KNN, Flann Based Matching and Ratio Based Matching to get more accurate results. Moreover I have used a dataset having various variations like change in illumination of images, rotation, scale, clarity etc. Using this dataset, I have found the number of matches between each set of image and the time taken for matching the images. After that, there is a RANSAC algorithm which uses homography matrix. It is basically used to eliminate faulty matching points (noise). I have implemented ORB with RANSAC and applied the same dataset and got improved results with less execution time.

3. ORB IMPLEMENTATION

Technology Used : C++, OpenCV, Codeblocks

Hence implementation can be divided into two steps as described below :

3.1. KEYPOINTS DETECTION

Keypoints are the same as interest points. These are special points in the image which differentiates the image and make it stand out.

Here are few screenshots of the output of keypoint detection in ORB :

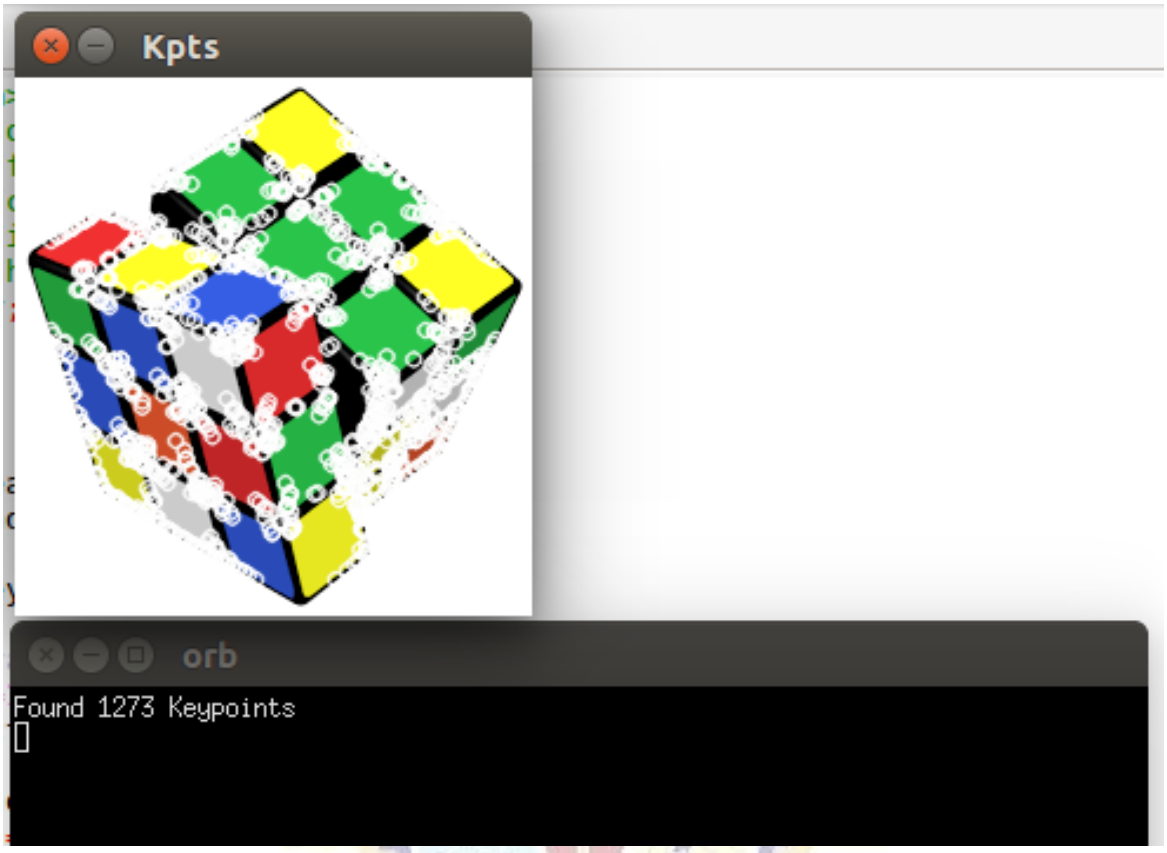


Fig. 2 Keypoints Detection - 1

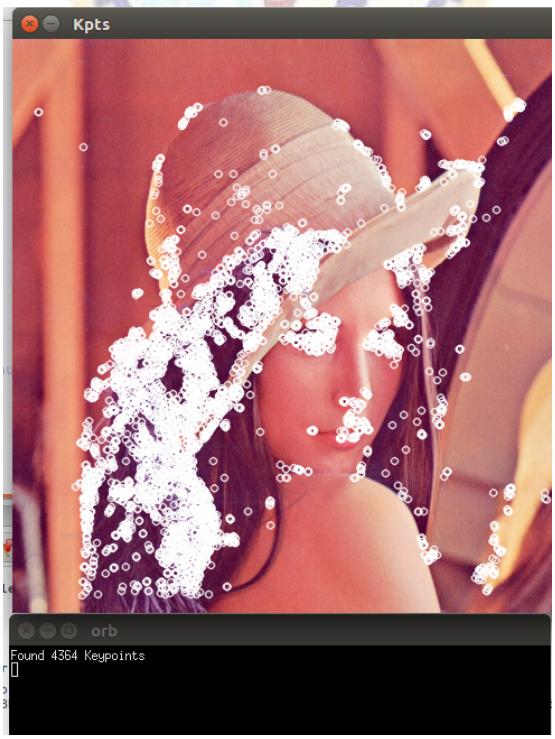


Fig. 3 Keypoints Detection – 2

Various arguments like threshold of edges, score measure type, number of levels, number of features etc. need to be specified.

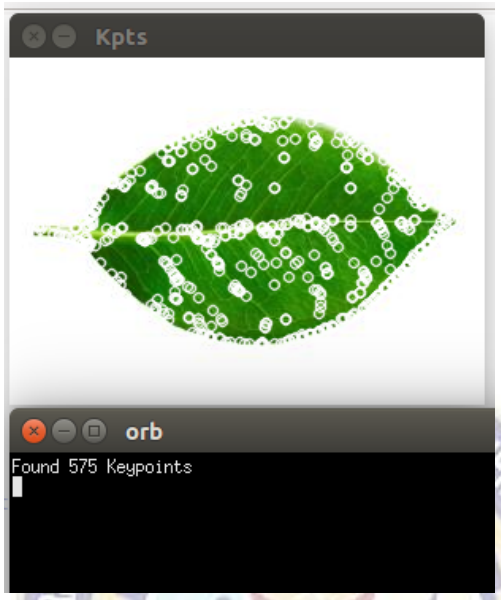


Fig. 4 Keypoints Detection – 3

3.2. IMAGES MATCHING

Descriptors generated using ORB can be the further used to match images

For each of the methods, first of all, images are being converted into grayscale.

3.2.1. Brute Force Matching

Brute-Force matcher uses distance calculator and takes the descriptor of one feature in first set and is matched with all the features in second set.

But the result is so many matches and lots of noise. In the paper [3], Brute Force Technique has been applied.

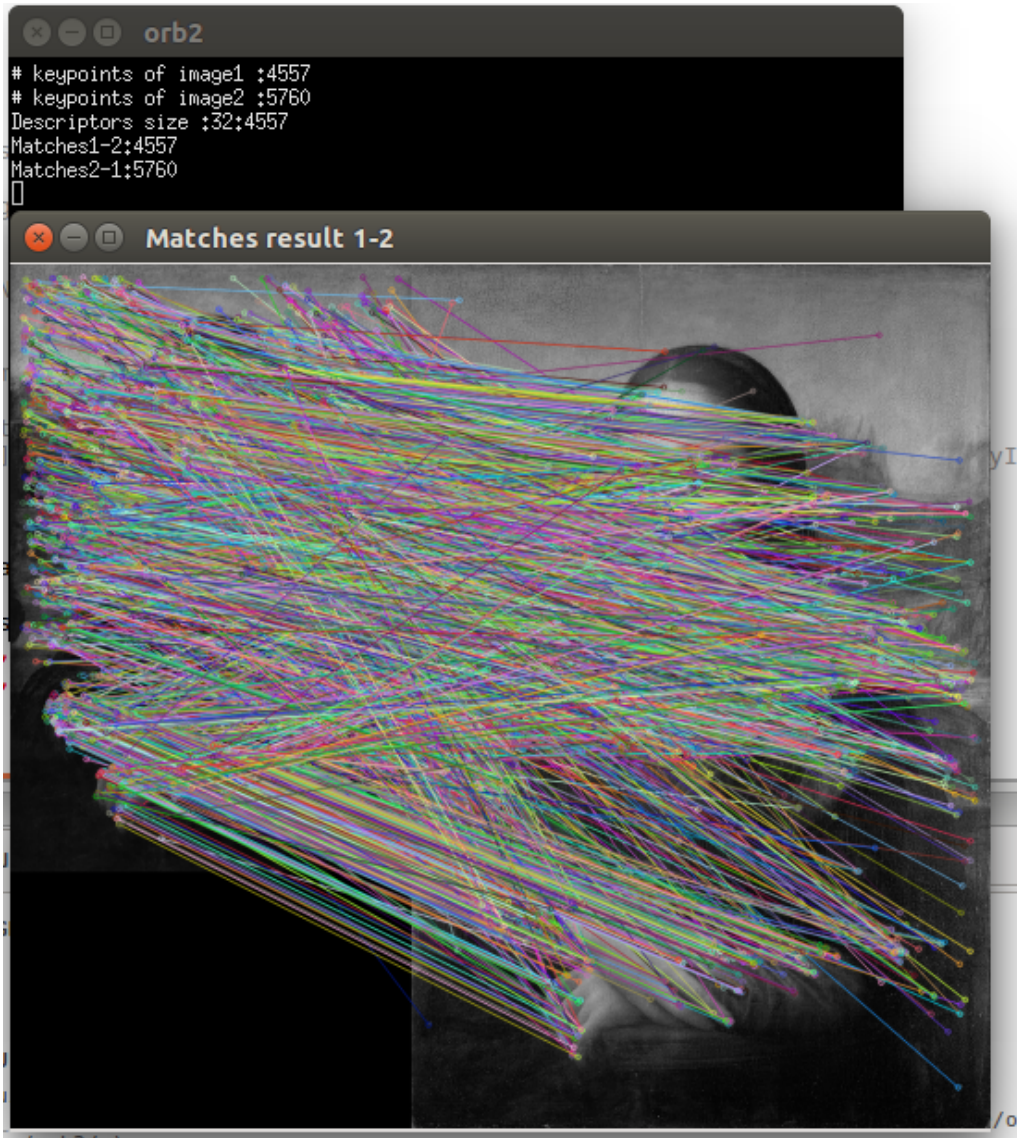


Fig. 5 Brute Force Matching

3.2.2. Match By Filtering Distance

Matches are filtered using distance after applying bruteforce method. Knn match is being done. According to the value of k, k best matches are shown.

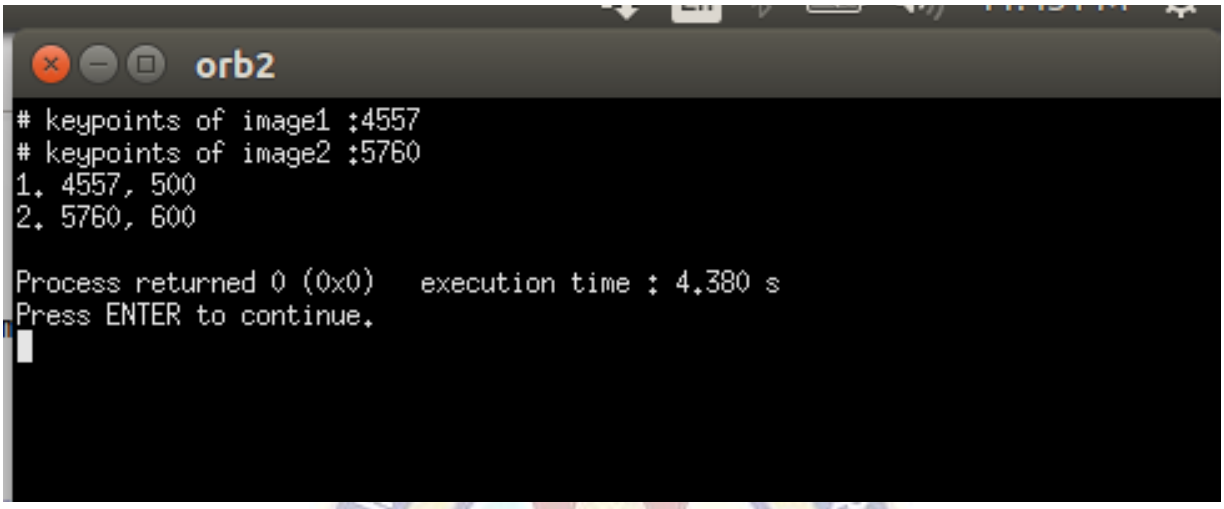
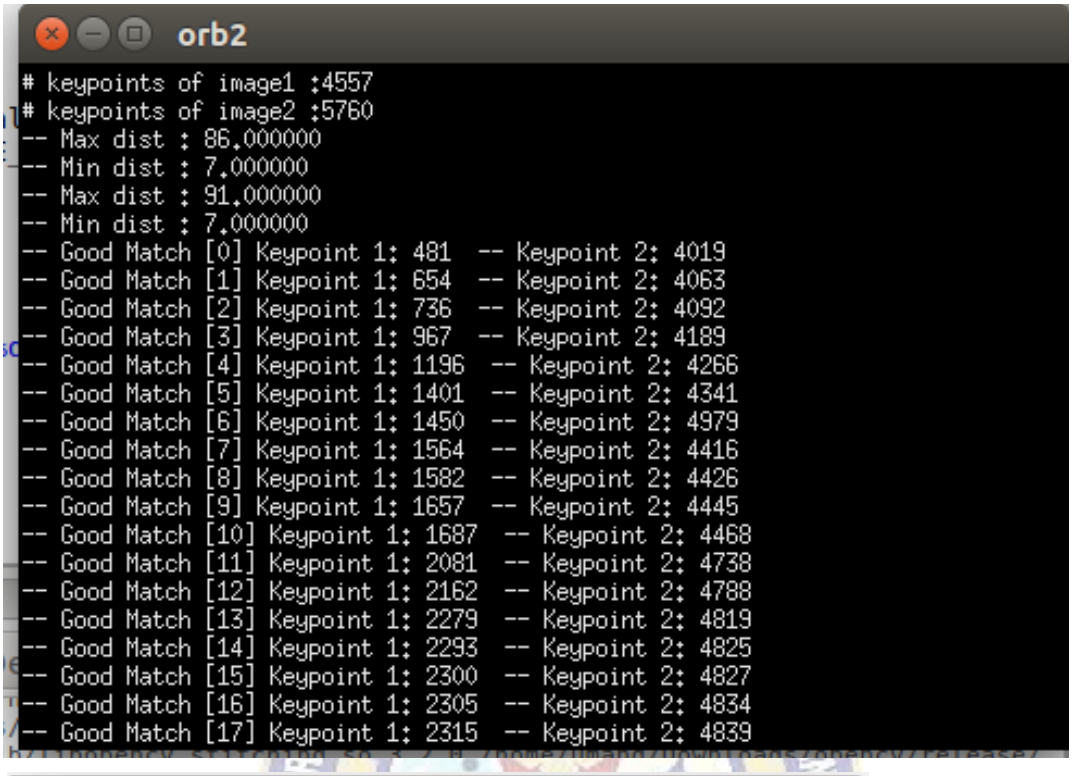


Fig. 6 Match by Filtering Distance

Here we chose k value to be 500 for image 1 and 600 for image 2. Out of all, only k best matches are chosen

3.2.3. Flann Based Matching

FLANN is an ensemble of various algorithms optimized for high dimensional features and for fast nearest neighbor search in huge datasets.



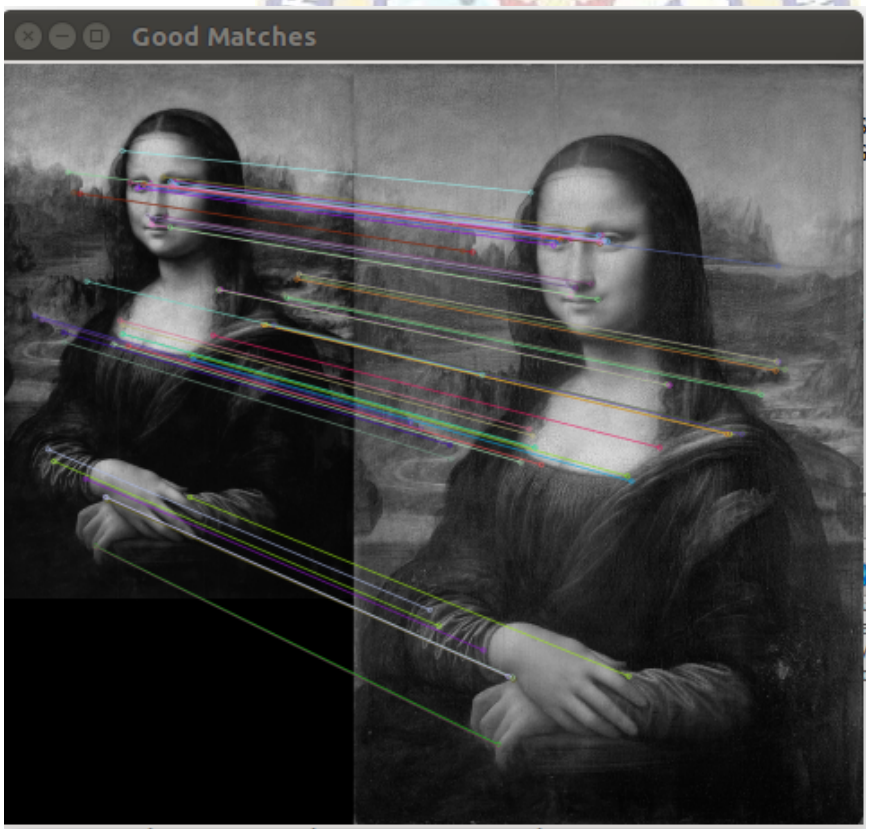


Fig. 7 Flann Based Matching

3.2.4. Ratio Test Using KNN

It is based on the ratio between first and second neighbor distance.

- When ratio is kept as 0.5

Fig. 8 Ratio Test using KNN –: Ratio=0.5

- When ratio is kept as 1

Fig. 9 Ratio Test using KNN –: Ratio=1

3.3 ORB Results On Dataset

Implementation of ORB has been tested against dataset having variation of scale, brightness, clarity and rotation of images with variations in the amount of change. Below is the result on the dataset -

Image		Bruteforce Matching			Match by filtering distance		Flann-based matching		Ratio-based matching			
									Ratio=0.8		Ratio=0.5	
		Time Taken	Matches1-2	Matches2-1	K=500	K=100	Time Taken	Matches	Time Taken	Matches	Time Taken	Matches
BARK	bimg1.pgm	3.87349	8272	8273	24.7943	5.20892	3.8776	934	4.09153	6909	4.06989	6582
	bimg1.png	3.85069	8272	8289	24.8272	5.20645	3.84258	165	4.0732	6904	4.06192	6567
	bimg1.ppm	3.86041	8272	8273	25.0112	5.23366	3.85761	934	4.05673	6909	4.04075	6582
	bimg2.jpeg	3.0431	8272	6556	21.4517	4.23247	3.07782	208	3.08287	439	3.06096	20
	bimg2.pgm	3.04254	8272	6529	21.4453	4.21265	3.02997	263	3.06239	411	3.05013	17
	bimg2.png	3.0517	8272	6552	21.3075	4.1983	3.05089	280	3.10163	440	3.06227	19
	bimg2.ppm	3.03583	8272	6530	21.4624	4.23897	3.06561	180	3.04815	409	3.07758	17
	bimg3.jpeg	3.85713	8272	8273	25.0116	5.22498	3.86208	55	3.86791	476	3.83271	49
	bimg3.pgm	3.87115	8272	8264	24.9368	5.17701	3.8536	50	3.88992	479	3.90493	65
	bimg3.png	3.84727	8272	8274	24.9956	5.2039	4.05895	97	3.8925	474	3.84407	60
	bimg3.ppm	3.86301	8272	8265	24.8456	5.16595	3.86712	71	3.88863	481	3.86472	65
	bimg4.jpeg	4.28948	8272	9196	26.8408	5.73149	4.30433	192	4.31462	252	4.29689	9
	bimg4.pgm	4.29557	8272	9202	26.8606	5.69485	4.27976	205	4.29672	237	4.31708	11
	bimg4.png	4.28674	8272	9195	26.4761	5.69112	4.29558	195	4.31548	239	4.29071	13
	bimg4.ppm	4.33922	8272	9202	26.6688	5.68738	4.29882	205	4.32541	237	4.26923	11
	bimg5.jpeg	4.23543	8272	8987	26.0335	5.66238	4.23235	17	4.21656	395	4.30269	396
	bimg5.pgm	4.17756	8272	8953	25.9909	5.63474	4.21123	32	4.20233	399	4.27698	402
	bimg5.png	4.22325	8272	8961	26.065	5.65235	4.19944	43	4.21842	402	4.26003	435
	bimg5.ppm	4.22372	8272	8954	26.0521	5.64426	4.21694	32	4.232	399	4.27283	402
	bimg6.jpeg	4.3265	8272	9227	26.6297	5.84232	4.3311	50	4.34554	167	4.42474	294
	bimg6.pgm	4.33143	8272	9218	26.5545	5.77427	4.32001	49	4.33694	174	4.39848	303
	bimg6.png	4.34613	8272	9238	26.6245	5.79989	4.31413	47	4.34221	170	4.42441	294
	bimg6.ppm	4.33833	8272	9218	26.5489	5.8029	4.31518	49	4.3476	174	4.35787	303
BIKES	biimg1.pgm	4.11736	8558	8539	26.372	5.58662	4.10999	1228	4.36875	7479	4.93497	21606
	biimg1.png	4.13189	8558	8544	26.3354	5.56955	4.11759	228	4.39161	7429	4.91371	21351
	biimg1.ppm	4.12559	8558	8539	25.9173	5.57221	4.12264	1228	4.38165	7479	5.00117	21606
	biimg2.jpeg	1.75658	8558	3648	15.3425	2.70858	1.75451	29	1.80767	2572	1.87939	4635
	biimg2.pgm	1.74554	8558	3653	15.3771	2.70653	1.7644	22	1.80847	2573	1.8891	4638
	biimg2.png	1.79331	8558	3722	15.6383	2.73456	1.78744	25	1.84709	2600	1.91291	4701
	biimg2.ppm	1.77918	8558	3653	15.1193	2.70931	1.76801	22	1.8422	2573	1.91202	4638
	biimg3.jpeg	1.32937	8558	2740	13.166	2.14314	1.33205	23	1.35669	1839	1.39805	2967
	biimg3.pgm	1.32698	8558	2754	13.1919	2.15497	1.32426	4	1.36448	1857	1.41099	2970
	biimg3.png	1.34115	8558	2777	13.1032	2.16132	1.33931	19	1.36901	1831	1.41505	2976
	biimg3.ppm	1.32953	8558	2754	12.99744	2.16068	1.33079	4	1.35191	1857	1.40366	2970
	biimg4.jpeg	0.796893	8558	1653	9.51327	1.44985	0.797541	37	0.802802	949	0.852917	1344
	biimg4.pgm	0.784485	8558	1634	9.71519	1.43938	0.791118	20	0.788596	944	0.851933	1314
	biimg4.png	0.795817	8558	1646	9.80643	1.44463	0.794385	19	0.803897	939	0.841193	1323
	biimg4.ppm	0.785912	8558	1634	9.7944	1.44728	0.791033	20	0.803174	944	0.83815	1314
	biimg5.jpeg	0.530556	8558	1102	7.93166	1.10577	0.531077	5	0.535517	631	0.568652	837
	biimg5.pgm	0.541952	8558	1116	8.00263	1.10757	0.542314	4	0.550632	635	0.579041	813
	biimg5.png	0.539341	8558	1121	8.00224	1.11696	0.540721	12	0.550642	631	0.570117	801
	biimg5.ppm	0.541179	8558	1118	7.95445	1.10498	0.539193	4	0.549487	635	0.577952	813
	biimg6.jpeg	0.358676	8558	744	6.27065	0.853803	0.360941	9	0.363868	378	0.393594	405
	biimg6.pgm	0.361204	8558	746	6.30302	0.86197	0.364113	12	0.367978	370	0.396142	399
	biimg6.png	0.350342	8558	735	6.17896	0.850528	0.356585	12	0.364467	350	0.384251	396
	biimg6.ppm	0.360578	8558	746	6.25809	0.849539	0.363075	12	0.364687	370	0.385094	399
BOAT	boimg1.pgm	34.4152	24639	24573	113.46	39.3416	34.5302	6091	36.4088	21971	40.5536	63792
	boimg1.png	34.229	24639	24573	113.764	39.3818	34.5945	6091	36.8051	21971	40.6505	63792
	boimg1.ppm	34.4295	24639	24573	113.901	39.2689	34.3695	6091	36.5245	21971	40.5583	63792
	boimg2.jpeg	35.5899	24639	25470	117.191	40.4061	35.5748	208	35.6859	3209	35.7253	438
	boimg2.pgm	35.6497	24639	25446	117.403	40.5862	35.5617	201	35.5716	3227	35.7141	441
	boimg2.png	35.5557	24639	25446	117.427	40.5764	35.4734	201	35.6348	3227	35.6061	441
	boimg2.ppm	35.5049	24639	25446	117.281	40.5479	35.4659	201	35.6439	3227	35.6805	441
	boimg3.jpeg	32.2284	24639	22999	108.979	36.8136	32.1013	302	32.3029	5732	32.3764	5874
	boimg3.pgm	32.1023	24639	22971	108.844	36.6906	32.0733	301	32.2334	5736	32.2696	5931
	boimg3.png	31.9976	24639	22971	108.929	36.7327	31.9733	301	32.3102	5736	32.3193	5931
	boimg3.ppm	32.1237	24639	22971	108.937	36.8197	32.1118	301	32.2667	5736	32.3472	5931
	boimg4.jpeg	23.3242	24639	16685	87.2899	27.2581	23.2572	225	23.3289	1613	23.2622	774
	boimg4.pgm	22.7951	24639	16331	86.0796	26.7605	22.7688	337	22.8592	1603	22.9795	771
	boimg4.png	22.7758	24639	16331	85.9682	26.6916	22.7961	337	22.8751	1603	22.9336	771
	boimg4.ppm	22.7619	24639	16331	86.0777	26.7291	22.8122	337	22.7852	1603	22.9129	771
	boimg5.jpeg	17.8784	24639	12792	74.0494	21.357	17.7845	372	17.845	365	17.98	21
	boimg5.pgm	17.7551	24639	12743	73.712	21.1423	17.7083	380	17.8092	367	17.8932	27
	boimg5.png	17.6983	24639	12743	73.7583	21.2652	17.7548	380	17.749	367	17.8504	27
	boimg5.ppm	17.7449	24639	12743	73.7608	21.3027	17.6998	380	17.7262	367	17.8598	27
	boimg6.jpeg	21.7974	24639	15673	83.2006	25.6442	21.8413	23	21.8171	395	21.9613	87
	boimg6.pgm	21.8391	24639	15663	83.157	25.6902	21.8343	28	21.8127	389	21.9834	84
	boimg6.png	21.8082	24639	15663	83.2292	25.6533	21.8872	28	21.769	389	21.9766	84
	boimg6.ppm	21.8853	24639	15663	82.9952	25.6716	21.825	28	21.8939	389	21.9347	84
GRAF	gimg1.pgm	3.01837	7313	7284	20.8962	4.15259	3.00283	1836	3.17882	6507	3.58994	18927
	gimg1.png	2.99703	7313	7250	20.9196	4.1367	2.98161	617	3.17452	6447	3.55357	18762
	gimg1.ppm	3.00893	7313	7285	21.0415	4.15779	3.0164	1845	3.19389	6508	3.58672	18930
	gimg2.jpeg	3.68628	7313	8903	24.3754	4.98927	3.67016	10	3.69485	1217	3.74504	510
	gimg2.pgm	3.64405	7313	8832	24.1933	4.96718	3.64723	27	3.69471	1243	3.69426	474
	gimg2.png	3.6498	7313	8841	24.0725	4.92768	3.64475	7	3.67809	1219	3.69294	480
	gimg2.ppm	3.65058	7313	8831	24.1742	4.95445	3.64745	27	3.65471	1242	3.69389	471
	gimg3.jpeg	4.12066	7313	10006	26.2953	5.56003	4.12643	51	4.14602	372	4.17502	45
	gimg3.pgm	4.11181	7313	9921	26.1652	5.49614	4.08537	22	4.12035	376	4.13924	39
	gimg3.png	4.08672	7313	9905	26.0489	5.48927	4.09039	22	4.08623	370	4.14674	39
	gimg3.ppm	4.11566	7313	9923	26.1381	5.50805	4.10413	21	4.09439	375	4.16197	39
	gimg4.jpeg	4.49671	7313	10871	28.0566	6.00512	4.49319	75	4.5062	116	4.54275	3
	gimg4.pgm	4.42619	7313	10712	27.6938	5.90827	4.4372	14	4.44117	110	4.48091	9

	gimg4.pgm	4.42619	7313	10712	27.6938	5.90827	4.4372	14	4.44117	110	4.48091	9
	gimg4.png	4.42134	7313	10717	27.6769	5.91557	4.42663	23	4.41951	110	4.48277	9
	gimg4.ppm	4.41678	7313	10709	27.7041	5.92026	4.41451	14	4.44754	110	4.46168	9
	gimg5.jpeg	4.40675	7313	10735	27.6187	5.93189	4.41136	26	4.45279	46	4.47794	0
	gimg5.pgm	4.36203	7313	10629	27.5608	5.86144	4.38083	46	4.40058	57	4.46411	0
	gimg5.png	4.40048	7313	10652	27.5301	5.87147	4.40335	44	4.40151	54	4.44928	0
	gimg5.ppm	4.34813	7313	10627	27.5716	5.8791	4.37741	46	4.38553	56	4.44288	0
	gimg6.jpeg	6.17595	7313	14991	35.4009	8.06818	6.17528	118	6.21233	52	6.26795	0
	gimg6.pgm	6.14418	7313	14731	34.7588	7.90011	6.08589	179	6.07386	51	6.17982	0
	gimg6.png	6.12228	7313	14751	34.9758	7.90586	6.06114	179	6.10881	48	6.15673	0
	gimg6.ppm	6.07953	7313	14729	34.9773	7.94159	6.10516	179	6.08362	51	6.12853	0
	img1.pgm	2.00973	10413	3432	17.854	3.10227	2.02625	14	2.02936	1455	2.07564	1419
	img1.png	1.99681	10413	3401	17.6552	3.08699	2.0034	28	2.02778	1452	2.05288	1422
	img1.ppm	2.00777	10413	3431	17.8082	3.10291	2.01549	14	2.04325	1455	2.07362	1410
	img2.jpeg	4.96649	10413	8466	31.1482	6.65142	4.96801	19	5.11426	4800	5.22257	7971
	img2.pgm	4.84507	10413	8243	30.5997	6.52488	4.84334	2	4.98595	4780	5.07474	7959
	img2.png	4.85279	10413	8233	30.6261	6.50395	4.84924	11	4.99768	4795	5.09624	8007
	img2.ppm	4.84886	10413	8241	30.684	6.53616	4.8448	2	4.98839	4778	5.0959	7965
	img3.jpeg	4.08667	10413	6969	27.5032	5.6236	4.09182	20	4.20153	3657	4.26374	5433
	img3.pgm	3.9992	10413	6840	27.0126	5.53481	4.01342	4	4.09675	3654	4.17387	5505
	img3.png	4.02181	10413	6843	27.0883	5.55061	4.02002	5	4.11725	3672	4.19091	5397
	img3.ppm	4.03007	10413	6840	27.1516	5.53788	4.0156	4	4.09825	3652	4.17892	5505
LEUVEN	img4.jpeg	3.42512	10413	5824	24.4809	4.80034	3.42155	3	3.486	2866	3.53295	3726
	img4.pgm	3.32777	10413	5676	24.118	4.69637	3.3279	4	3.39047	2835	3.45863	3738
	img4.png	3.30797	10413	5667	24.1414	4.68158	3.32023	30	3.37291	2858	3.44144	3816
	img4.ppm	3.34022	10413	5677	24.1014	4.6851	3.34221	4	3.39175	2838	3.44628	3747
	img5.jpeg	2.70241	10413	4599	21.2118	3.91624	2.69895	7	2.73618	2115	2.76914	2361
	img5.pgm	2.66159	10413	4532	20.9343	3.89878	2.67111	24	2.71437	2135	2.73652	2457
	img5.png	2.66125	10413	4529	21.0658	3.89818	2.65954	24	2.7019	2118	2.74094	2340
	img5.ppm	2.65712	10413	4532	21.0384	3.88787	2.65646	24	2.70279	2135	2.73332	2457
	img6.jpeg	2.07567	10413	3500	18.0479	3.18052	2.06002	13	2.07159	1460	2.12762	1443
	img6.pgm	2.01324	10413	3432	17.8695	3.11114	2.01565	14	2.0378	1455	2.07112	1419
	img6.png	1.98916	10413	3401	17.7521	3.07874	2.00449	28	2.02264	1452	2.05924	1422
	img6.ppm	2.02499	10413	3431	17.8869	3.10236	2.01847	14	2.04362	1455	2.07984	1410
	timg1.pgm	45.7902	28432	28426	139.348	51.5355	45.8538	5402	48.5018	26026	54.0092	76062
	timg1.png	45.8398	28432	28446	139.272	51.5739	45.8668	1732	48.6091	25957	54.1097	75858
	timg1.ppm	45.8332	28432	28426	139.247	51.492	45.8896	5402	48.5643	26026	53.9899	76062
	timg2.jpeg	45.9257	28432	28505	139.547	51.7059	45.8075	1	46.1828	5922	46.1983	4305
	timg2.pgm	45.9034	28432	28472	139.408	51.6058	45.8937	1	46.0578	5963	46.1794	4389
	timg2.png	45.9034	28432	28478	139.599	51.6369	45.7011	3	45.9883	5937	46.1583	4383
	timg2.ppm	45.8779	28432	28472	139.585	51.6216	45.7089	1	46.0511	5963	46.1584	4389
	timg3.jpeg	45.7271	28432	28430	138.884	51.5403	45.8298	43	46.9677	4738	46.0524	3003
	timg3.pgm	45.9211	28432	28451	139.3	51.6016	45.8585	43	46.0692	4737	46.057	2976
	timg3.png	45.914	28432	28439	139.391	51.5672	45.7769	15	45.9809	4702	46.11447	2970
	timg3.ppm	46.0151	28432	28451	139.353	51.6334	45.8488	43	46.0701	4737	46.0673	2976
TREES	timg4.jpeg	41.0509	28432	25461	127.839	46.356	40.9263	28	41.0739	2815	41.1391	1281
	timg4.pgm	40.8215	28432	25330	127.395	46.0682	40.7905	24	40.855	2767	40.9194	1251
	timg4.png	40.7703	28432	25336	127.384	46.1128	40.7654	21	40.8223	2761	41.03	1176
	timg4.ppm	40.8349	28432	25330	127.315	46.1828	40.8459	24	40.8949	2767	40.9495	1251
	timg5.jpeg	25.848	28432	16109	93.2512	30.0943	25.9415	19	25.9852	1554	26.0623	579
	timg5.pgm	25.9332	28432	16095	93.3205	30.0391	25.9644	20	25.9859	1531	26.045	588
	timg5.png	25.8946	28432	16104	93.4073	30.0182	25.9121	21	25.8971	1523	26.1416	594
	timg5.ppm	25.9286	28432	16095	93.3819	30.1285	25.9454	20	25.8399	1531	25.9642	588
	timg6.jpeg	16.1707	28432	10030	69.4986	19.4584	16.1443	72	16.1002	769	16.2838	177
	timg6.pgm	16.2176	28432	10048	69.6747	19.3674	16.2092	106	16.0906	779	16.2397	192
	timg6.png	16.186	28432	10055	69.6721	19.587	16.186	71	16.2038	753	16.3374	204
	timg6.ppm	16.1389	28432	10048	69.6953	19.4944	16.188	106	16.1607	779	16.1905	192
	uimg1.pgm	24.6094	20900	20805	89.6136	28.6026	24.5965	2726	26.0196	18265	28.6979	52836
	uimg1.png	24.5805	20900	20766	89.4542	28.6234	24.5765	635	25.9229	18153	28.5953	52383
	uimg1.ppm	24.6659	20900	20804	89.7966	28.6713	24.551	2717	26.0193	18266	28.7054	52842
	uimg2.jpeg	24.1777	20900	20466	88.7704	28.2213	24.2337	127	25.161	13670	26.1999	32250
	uimg2.pgm	24.2237	20900	20423	88.4558	28.1138	23.9537	124	25.0924	13670	26.1739	32379
	uimg2.png	24.1236	20900	20429	88.4468	28.2217	24.1933	80	24.9689	13641	26.0909	32196
	uimg2.ppm	24.0694	20900	20423	88.4328	28.1778	24.1876	124	25.0115	13670	26.0944	32379
	uimg3.jpeg	24.2772	20900	20512	89.0457	28.2978	24.3084	24	24.9553	11516	25.503	23004
	uimg3.pgm	24.2669	20900	20468	88.8504	28.2213	24.1903	22	24.9098	11503	25.4258	23049
	uimg3.png	24.2901	20900	20498	88.6602	28.1817	24.1138	19	24.9039	11551	25.3671	22911
	uimg3.ppm	24.2507	20900	20468	88.9282	28.2563	24.1932	22	24.893	11503	25.4086	23049
UBC	uimg4.jpeg	23.6806	20900	19987	87.328	27.5498	23.5977	2	23.9575	7935	24.0547	12453
	uimg4.pgm	23.6097	20900	19864	87.07	27.4545	23.4476	2	23.8864	7972	24.0325	12501
	uimg4.png	23.653	20900	19938	87.2883	27.5949	23.6065	4	23.9548	7932	24.0262	12516
	uimg4.ppm	23.5468	20900	19864	87.095	27.2573	23.5305	2	23.7531	7972	23.9978	12501
	uimg5.jpeg	18.9018	20900	16033	75.1694	22.5157	18.9485	2	19.0983	4048	19.1437	4146
	uimg5.pgm	18.8235	20900	15975	75.2365	22.4461	18.9575	12	18.9759	4031	19.0751	4065
	uimg5.png	18.9051	20900	15979	75.0986	22.3913	18.8813	1	19.0026	4008	19.0578	4080
	uimg5.ppm	18.9014	20900	15975	75.184	22.3635	18.9056	12	18.9901	4031	19.0051	4065
	uimg6.jpeg	11.4106	20900	9693	55.0883	14.1584	11.4418	27	11.4797	1707	11.5553	1152
	uimg6.pgm	11.3628	20900	9592	54.6835	14.0586	11.3562	27	11.3446	1700	11.4242	1170
	uimg6.png	11.3901	20900	9621	54.7785	14.0126	11.3734	27	11.3817	1695	11.468	1161
	uimg6.ppm	11.3588	20900	9592	54.6976	14.0072	11.3664	27	11.3581	1700	11.4522	1170
	wimg1.pgm	42.9583	27454	27424	131.432	48.1199	42.6978	2459	45.142	24634	50.1064	71667
	wimg1.png	42.7738	27454	27457	131.221	48.1831	42.7145	418	45.2266	24611	50.1394	71520
	wimg1.ppm	42.722	27454	27422	131.051	48.1344	42.5581	2446	45.0695	24630	50.2056	71664
	wimg2.jpeg	41.3351	27454	26585	127.629	46.6327	41.2708	16	41.7016	7070	41.6891	5499
	wimg2.pgm	41.3284	27454	26553	127.495	46.5637	41.3032	5	41.5265	7055	41.6219	5499

	wimg2.jpeg	41.3351	27454	26585	127.629	46.6327	41.2708	16	41.7016	7070	41.6891	5499
	wimg2.pgm	41.3284	27454	26553	127.495	46.5637	41.3032	5	41.5265	7055	41.6219	5499
	wimg2.png	41.2944	27454	26528	127.293	46.4783	41.1699	103	41.6286	7115	41.447	5571
	wimg2.ppm	41.2796	27454	26551	127.669	46.6237	41.2587	5	41.645	7059	41.5278	5493
	wimg3.jpeg	40.669	27454	26137	125.952	45.881	40.5704	2	40.7531	4095	40.7911	1890
	wimg3.pgm	40.5093	27454	26142	125.566	45.8552	40.6872	1	40.7155	4074	40.8983	1971
	wimg3.png	40.5711	27454	26134	125.897	45.8079	40.5835	28	40.7201	4026	40.7826	1902
	wimg3.ppm	41.1101	27454	26141	125.964	45.8014	40.5603	1	40.745	4069	40.782	1959
WALL	wimg4.jpeg	40.9896	27454	26329	127.402	46.216	41.1129	845	41.2143	1444	41.1713	153
	wimg4.pgm	40.8743	27454	26335	127.312	46.2349	40.8918	84	40.8167	1427	41.0795	153
	wimg4.png	40.9415	27454	26330	127.111	46.2839	41.0123	242	41.0563	1459	41.1239	171
	wimg4.ppm	40.9702	27454	26335	127.539	46.346	40.9566	84	40.9942	1426	41.0988	153
	wimg5.jpeg	41.0746	27454	26412	127.351	46.3538	41.2512	3585	41.0002	220	41.3292	0
	wimg5.pgm	40.908	27454	26355	127.542	46.2667	40.9298	2689	40.9869	216	41.1677	0
	wimg5.png	40.9941	27454	26350	127.213	46.324	41.0043	2662	41.0544	204	41.1933	3
	wimg5.ppm	41.0195	27454	26352	126.755	46.2508	41.0626	2695	40.9043	216	41.1269	0
	wimg6.jpeg	40.9314	27454	26250	127.37	46.1875	42.3418	6634	40.76	49	41.0329	0
	wimg6.pgm	40.7761	27454	26189	126.813	46.0396	42.1374	6635	40.6286	62	40.8482	0
	wimg6.png	40.7705	27454	26230	126.988	46.069	42.1818	6642	40.8902	72	41.0523	0
	wimg6.ppm	40.8736	27454	26191	126.678	46.0165	42.1948	6647	40.6747	60	40.9438	0

4. ORB IMPROVEMENT

To improve the time taken and quality of matches, I have used RANSAC and homography matrix to eliminate false matching points. RANSAC algorithm gives effective sample by calculating the mathematical model of the data parameter which contains abnormal data. For estimation of model, this algorithm uses less number of points and using the remaining points, it verifies the model.

After the comparison of keypoints descriptors given by ORB, we use it to find the homography matrix which is 3*3 matrix between two images. RANSAC algorithm firstly chooses 4 random matches assuming them to be inliers, then uses them to compute parameters of a homography matrix. This estimate created in previous step is then applied to all the other matches, using homography matrix, the keypoints in the target are projected only if their projection falls within a radius of the corresponding keypoint in the frame, accordingly they are then counted as inliers. This estimate is repeated N number of times, it returns the estimate homography matrix with the largest number of inliers. [8]

For each feature found in an image, this algorithm may find more than 1 candidate matching points in another image. It makes use of threshold to filter out some of the candidate matching points. The more is the number of good matching points selected, the more is the possibly that RANSAC will give a correct matrix.

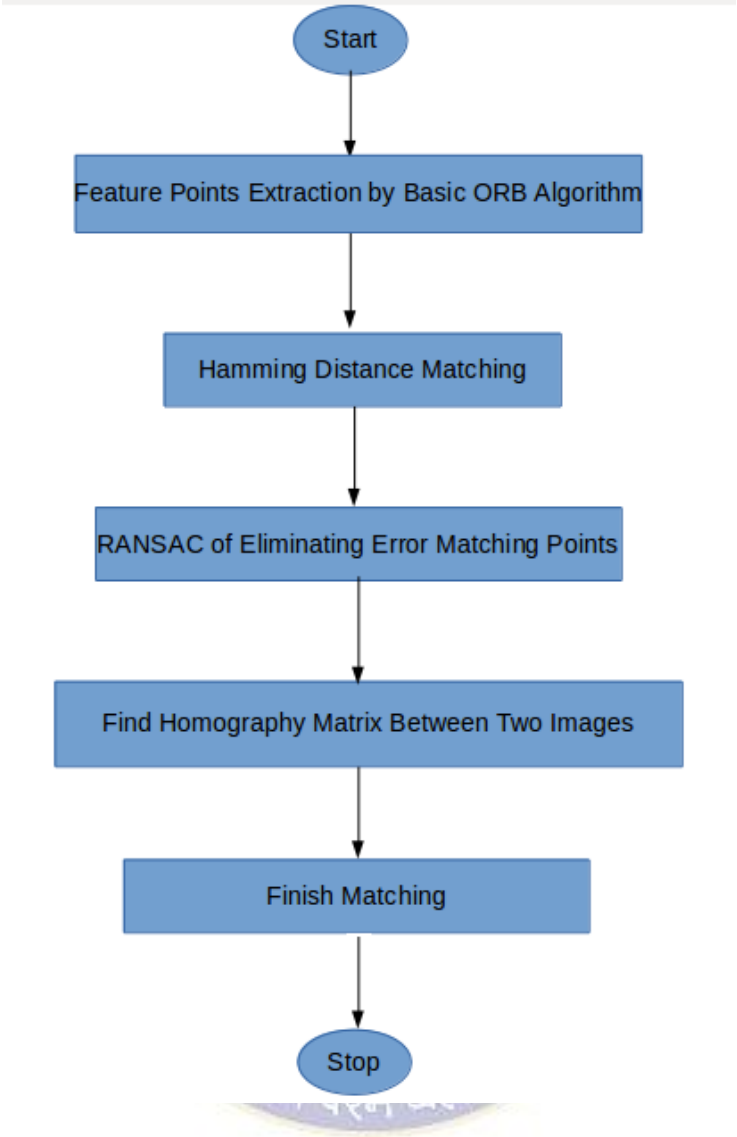


Fig. 10 Flowchart of Improved ORB Algorithm

Until feature points extraction, everything is the same as in the basic ORB algorithm, after that hamming distance matching is done between two images. RANSAC is then used to eliminate faulty or error matching points. Then Homography matrix is found between two images.

Below is the implementation result of improved ORB -

fig. 11 Improved ORB Implementation

4.1. Improved ORB Results on Dataset

On the same dataset which has been used for ORB, we have tried the improved version of ORB. Below are the results -

Image		RANSAC	
		Time Taken	Matches
BARK	bimg1.pgm	1.93149	7026
	bimg1.png	1.94802	7009
	bimg1.ppm	1.94001	7026
	bimg2.jpeg	1.55917	1042
	bimg2.pgm	1.58232	865
	bimg2.png	1.58677	932
	bimg2.ppm	1.58848	896
	bimg3.jpeg	1.99945	652
	bimg3.pgm	1.97981	737
	bimg3.png	1.99377	354
	bimg3.ppm	1.99418	335
	bimg4.jpeg	2.22504	35
	bimg4.pgm	2.22022	291
	bimg4.png	2.21162	49
	bimg4.ppm	2.2189	48
	bimg5.jpeg	2.14217	341
	bimg5.pgm	2.158	195
	bimg5.png	2.15196	366
	bimg5.ppm	2.15735	70
	bimg6.jpeg	2.20844	11
	bimg6.pgm	2.22666	12
	bimg6.png	2.21885	12
	bimg6.ppm	2.20198	14
BIKES	biimg1.pgm	2.07977	7579
	biimg1.png	2.06647	7534
	biimg1.ppm	2.0813	7579
	biimg2.jpeg	0.886564	2281
	biimg2.pgm	0.887598	2122
	biimg2.png	0.906457	2516
	biimg2.ppm	0.882344	2122
	biimg3.jpeg	0.663021	1427
	biimg3.pgm	0.668933	1648
	biimg3.png	0.670727	1687
	biimg3.ppm	0.663466	1648
	biimg4.jpeg	0.404084	778
	biimg4.pgm	0.397668	721
	biimg5.pgm	0.271513	555
	biimg5.png	0.280421	381
	biimg5.ppm	0.26737	555
	biimg6.jpeg	0.184171	252
	biimg6.pgm	0.188747	247
	biimg6.png	0.189633	217
	biimg6.ppm	0.186071	247
BOAT	boimg1.pgm	2.83691	9389
	boimg1.png	2.84015	9389
	boimg1.ppm	2.84474	9389
	boimg2.jpeg	2.83889	2874
	boimg2.pgm	2.83167	3075
	boimg2.png	2.84452	3075
	boimg2.ppm	2.82441	3075
	boimg3.jpeg	2.83078	3292
	boimg3.pgm	2.83401	2730
	boimg3.png	2.82108	2730
	boimg3.ppm	2.83404	2730
	boimg4.jpeg	2.70917	1549
	boimg4.pgm	2.70084	1703

	boimg4.pgm	2.70084	1703
	boimg4.png	2.69678	1703
	boimg4.ppm	2.70999	1703
	boimg5.jpeg	2.70844	861
	boimg5.pgm	2.6796	818
	boimg5.png	2.68764	818
	boimg5.ppm	2.69162	818
	boimg6.jpeg	2.66561	36
	boimg6.pgm	2.67181	240
	boimg6.png	2.67188	240
	boimg6.ppm	2.66887	240
GRAF	gimg1.pgm	1.51639	6569
	gimg1.png	1.49583	6516
	gimg1.ppm	1.50749	6570
	gimg2.jpeg	1.8464	2194
	gimg2.pgm	1.81699	1877
	gimg2.png	1.84098	1908
	gimg2.ppm	1.82307	2036
	gimg3.jpeg	2.08471	975
	gimg3.pgm	2.07395	783
	gimg3.png	2.07029	962
	gimg3.ppm	2.05327	929
	gimg4.jpeg	2.10555	21
	gimg4.pgm	2.1029	38
	gimg4.png	2.08643	35
	gimg4.ppm	2.09596	37
	gimg5.jpeg	2.09926	9
	gimg5.pgm	2.0892	7
	gimg5.png	2.10878	8
	gimg5.ppm	2.09692	8
	gimg6.jpeg	2.11125	8
	gimg6.pgm	2.12219	8
	gimg6.png	2.10597	8
	gimg6.ppm	2.09982	8
LEUVEN	img1.pgm	2.11566	7871
	img1.png	2.11805	7829
	img1.ppm	2.1105	7870
	img2.jpeg	1.91216	4729
	img2.pgm	1.89383	4393
	img2.png	1.89131	4609
	img2.ppm	1.89504	4817
	img3.jpeg	1.72154	3267
	img3.pgm	1.67294	3892
	img3.png	1.67238	3470
	img3.ppm	1.67098	3750
	img4.jpeg	1.40706	3041
	img4.pgm	1.37959	3014
	img4.png	1.38676	3193
	img4.ppm	1.38996	3132
	img5.jpeg	1.12402	2162
	img5.pgm	1.10983	2404
	img5.png	1.10699	2388
	img5.ppm	1.09471	2216
	img6.jpeg	0.856712	1651
	img6.pgm	0.840795	1437
	img6.png	0.830806	1649
	img6.ppm	0.843118	1780
TREES	timg1.pgm	2.85215	9384
	timg1.png	2.83706	9349
	timg1.ppm	2.84445	9369
	timg2.jpeg	2.82728	2753
	timg2.pgm	2.80629	2424
	timg2.png	2.82528	2407
	timg2.ppm	2.82573	2485
	timg3.jpeg	2.84165	2354
	timg3.pgm	2.84137	2140
	timg3.png	2.81961	2125
	timg3.ppm	2.81806	2201
	timg4.jpeg	2.89815	1481
	timg4.pgm	2.87291	1399
	timg4.png	2.87622	1499
	timg4.ppm	2.86444	1399
	timg5.jpeg	2.87223	983
	timg5.pgm	2.89922	954
	timg5.png	2.89943	1063
	timg5.ppm	2.88196	954
	timg6.jpeg	2.52331	355
	timg6.pgm	2.53217	480
	timg6.png	2.53497	349
	timg6.ppm	2.52976	422

UBC	uimg1.pgm	2.64669	8954
	uimg1.png	2.6248	8930
	uimg1.ppm	2.65837	8955
	uimg2.jpeg	2.64363	7659
	uimg2.pgm	2.63829	7643
	uimg2.png	2.64033	7648
	uimg2.ppm	2.64212	7643
	uimg3.jpeg	2.63464	6971
	uimg3.pgm	2.62995	6955
	uimg3.png	2.64978	6993
	uimg3.ppm	2.64309	6955
	uimg4.jpeg	2.65422	5931
	uimg4.pgm	2.63207	6035
	uimg4.png	2.62988	6002
	uimg4.ppm	2.65236	6035
	uimg5.jpeg	2.66546	4423
	uimg5.pgm	2.65263	4223
	uimg5.png	2.67071	4069
	uimg5.ppm	2.65217	4223
	uimg6.jpeg	2.56248	2558
	uimg6.pgm	2.53854	2559
	uimg6.png	2.52571	2556
	uimg6.ppm	2.53592	2559
WALL	wimg1.pgm	2.83489	9160
	wimg1.png	2.82522	9139
	wimg1.ppm	2.81141	9159
	wimg2.jpeg	2.80666	3855
	wimg2.pgm	2.82156	3827
	wimg2.png	2.8258	3805
	wimg2.ppm	2.84055	3799
	wimg3.jpeg	2.82869	2656
	wimg3.pgm	2.81771	2868
	wimg3.png	2.81647	2843
	wimg3.ppm	2.83191	2660
	wimg4.jpeg	2.83072	1489
	wimg4.pgm	2.85484	1508
	wimg4.png	2.8595	1562
	wimg4.ppm	2.86227	1506
	wimg5.jpeg	2.85637	138
	wimg5.pgm	2.88806	215
	wimg5.png	2.90824	351
	wimg5.ppm	2.87249	57
	wimg6.jpeg	2.8905	9
	wimg6.pgm	2.87916	8
	wimg6.png	2.89684	7
	wimg6.ppm	2.87354	10

5. CONCLUSION

ORB is a combination of o-FAST and R-BRIEF, the improved versions of FAST and BRIEF algorithms. It performs much better than SIFT or SURF as seen in paper[5]. Though it is around two times faster than SIFT and SURF, but still there is a huge scope of improvement. ORB descriptors comparison can be done in the same way as done before, after that we can apply RANSAC and homography matrix to eliminate false and error matching points. Moreover as seen with the implementation of improved ORB and using the same dataset having variations of scale, brightness, clarity, rotation on both the original and improved ORB, we get a significant difference in the time taken by them.

6. FUTURE WORK

Future work that can be done to further improve the image matching and time taken to match images include Perspective Transformation and Median Filtering Method that can be done to Eliminate Noise and improve the image matching quality. Median filtering is used to remove the texture and keep edge sharpness in the same area, such as the trees in the background.

7. LEARNING OUTCOME

At the end of this course, I have learnt a lot. I have summarized my learnings as below:

- I have learnt about various latest keypoint and feature detection algorithms, what are the shortcomings of each of them and how they have been tackled in the upcoming algorithms.
- I have learnt how to build GUI in C++.
- I have learnt about how to use OpenCV with C++ under Ubuntu environment.
- I have learnt about various image matching approaches like brute force matching, distance based matching, flann based matching, ratio based matching.
- I have also learnt about the latest advancements going on in this area i.e. I have learnt about RANSAC and homography matrix and incorporated that in my code to improve the matching and time taken for matching.
- I have got a strong foundation to work on similar projects in the future.
- I have gained experience on how to write a research report.

8. REFERENCES

[1] Stefan Leutenegger, Margarita Chli and Roland Y. Siegwart; BRISK: Binary Robust invariant scalable keypoints; IEEE International Conference on Computer Vision; 2011

[2] Lisha Guo, Junshan Li, YingHong Zhu and ZongQi Tang; A novel Features from Accelerated Segment Test algorithm based on LBP on image matching; IEEE; 2011

[3] P M Panchal , S R Panchal and S K Shah; A Comparison of SIFT and SURF; International Journal of Innovative Research in Computer and Communication Engineering Vol. 1, Issue 2, April 2013

[4] PrashantAglave and Vijaykumar S. Kolkure; Comparative Study of Different Image Feature Extraction Algorithm and Representation Techniques; Asian Journal of Convergence in Technology Volume1, Issue 5; 2015

[5] Aomei Li , Wanli Jiang, Weihua Yuan, Dehui Dai, Siyu Zhang and Zhe Wei; An Improved FAST+SURF Fast Matching Algorithm; International Congress of Information and Communication Technology (ICICT); 2017

[6] Ethan Rublee, Vincent Rabaud, Kurt Konolige and Gary Bradski; ORB: an efficient alternative to SIFT or SURF ; IEEE International Conference on Computer Vision; 2011

[7] I-BRIEF: A Fast Feature Point Descriptor with More Robust Features; International Conference on Signal Image Technology & Internet-Based Systems; 2011

[8] Lei Yu , Zhixin Yu and Yan Gong; An Improved ORB Algorithm of Extracting and Matching Features*;* International Journal of Signal Processing, Image Processing and Pattern Recognition Vol. 8, No. 5, pp. 117-126, 2015

[9] Xiaowei Li, Yue Liu, Yongtian Wang and Dayuan Yan; “Computing Homography with RANSAC Algorithm”; SPIE Conference Proceedings; 2005

APPENDIX – 1 (SOURCE CODE)

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
//#include <opencv2/nonfree/features2d.hpp> //
//#include <opencv2/nonfree/nonfree.hpp>
#include <iostream>
#include <opencv2/opencv.hpp>
#include <opencv2/features2d.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <dirent.h>
#include <ctime>
using namespace cv;
using namespace std;
std::vector<DMatch> filter_distance(Mat descriptors,std::vector< DMatch > matches);
int main(int argc, const char *argv[])
{
   int ch;
   if(argc != 3)
   {
      cout << "usage:match <image1> <image2>\n" ;
      exit(-1);
   }
   string img1=string(argv[1]), img2 = string(argv[2]);
   Mat image1 = imread(img1, CV_LOAD_IMAGE_GRAYSCALE );
   Mat image2 = imread(img2, CV_LOAD_IMAGE_GRAYSCALE );
   if( !image1.data || !image2.data )
{
```

```
    std::cout<< " --(!) Error reading images " << std::endl; return -1;
  }
  std::vector<KeyPoint> kp,kp2;

  Ptr<ORB> extractor=ORB::create(features,scale,levels,edge,frst,WTA_K,score,patch,thresh);
  detector->detect(img, kp);
  detector->detect(imge2,kp2);

  cout << "# keypoints of image1 :" << kp.size() << endl;
  cout << "# keypoints of image2 :" << kp2.size() << endl;
  Mat desc1,desc2;
  extractor->compute(img,kp,desc1);
  extractor->compute(imge2,kp2,desc2);
  Ptr<DescriptorMatcher> matcher = DescriptorMatcher::create("BruteForceHamming");
  While(1)
    {
    cout<<"Enter 1 for Bruteforce match\nEnter 2 for match by filtering distance\nEnter 3 for Flann-Based matching\nEnter 4 for ratio based matching\nEnter 5 for improved ORB\nEnter Any other key to exit\n";
    cin>>ch;
    switch(ch)
    {
    case 1:
    {
      std::vector< DMatch > matches, matches2;
      clock_t begin = clock();
      matcher->match( desc, desc2, matches);
      matcher->match( desc2, desc, matches2);
```

```
//-- Draw matches
        clock_t end = clock();
        double elapsed_secs = double(end - begin) / CLOCKS_PER_SEC;
        cout << "Time Costs : " << elapsed_secs << endl;
        Mat img_matches, img_matches2;
        drawMatches( img, kp, imge2, kp2,matches, img_matches, Scalar::all(-1), Scalar::all(-1),
          vector<char>(), DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS );
        //drawMatches( img1, kp, imge2, kp2,matches2, img_matches2, Scalar::all(-1), Scalar::all(-1),
         // vector<char>(), DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS );
        cout << "Matches12:" << matches.size() << endl;
        cout << "Matches21:" << matches2.size() <<endl;
        namedWindow("Matches result 1-2",WINDOW_KEEPRATIO);
        imshow("Matches result 1-2",img_matches);
        //imshow("Matches result 2-1", img_matches2);
        waitKey(0);
        destroyAllWindows();
        break;
    }
    case 2:
    {
        int t;
        cin>>t;
        vector< vector<DMatch> > matchesx, matchesz;
        clock_t begin = clock();
        matcher->knnMatch( desc, desc2, matchesx,t );
        matcher->knnMatch( desc2, desc, matchesz, t);
        cout << "Matches 1 to 2 ==" << matchesx.size() << ", "<< matchesx[0].size() << endl;
```

```
    cout << "Matches 2 to 1 ==" << matchesz.size() << ", "<< matchesz[0].size() << endl;
    clock_t end = clock();
    double elapsed_secs = double(end - begin) / CLOCKS_PER_SEC;
    cout << "Time Costs : " << elapsed_secs << endl;
 //   Mat img_matches,img_matches2;


   /* drawMatches(img1,kp,imge2,kp2,matchesx[0],img_matches);
    imshow("Matches",img_matches);
    drawMatches(image1,kp,imge2,kp2,matchesz[0],img_matches2);
    imshow("Matches2",img_matches2);*/
    waitKey(0);
  //  destroyAllWindows();
    break;}
case 3:
{
    std::vector< DMatch > matches, matches2;
    clock_t begin = clock();
    matcher->match( desc1, desc2, matches );
    matcher->match( desc2, desc1, matches2 );
    std::vector< DMatch > goodmatches1, goodmatches2, bettermatches;
    goodmatches1 = filter_distance(desc1, matches);
    goodmatches2 = filter_distance(desc2, matches2);
    for(int i=0; i<goodmatches1.size(); i++)
     {
       DMatch temp1 = goodmatches1[i];
       for(int j=0; j<goodmatches2.size(); j++)
        {
```

```
            DMatch temp2 = goodmatches2[j];
            if(temp1.queryIdx == temp2.trainIdx && temp2.queryIdx == temp1.trainIdx)
            {
               bettermatches.push_back(temp1);
               break;
            }
         }
      }
      clock_t end = clock();
      double elapsed_secs = double(end - begin) / CLOCKS_PER_SEC;
      cout << "Time Costs : " << elapsed_secs << endl;
      //-- Draw only "good" matches
      Mat img_matches;
      drawMatches( img1, kp, imge2, kp2,
        better_matches, img_matches, Scalar::all(-1), Scalar::all(-1),
        vector<char>(), DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS );
      //-- Show detected matches
      namedWindow("Good matches",WINDOW_KEEPRATIO);
      imshow( "Good Matches", img_matches );
      //for( int i = 0; i < (int)bettermatches.size(); i++ )
       {
        //  printf( "-- Good Match [%d] Keypoint 1: %d  -- Keypoint 2: %d  \n", i, better_matches[i].queryIdx,
better_matches[i].trainIdx );
      //}
      cout << "Matches:" << better_matches.size() <<endl;
      waitKey(0);
      destroyAllWindows();
```

```
            break;
        }
        case 4:
        {
            vector< vector<DMatch> > matchesx, matchesz;
            double ratio;
            cout<<"Enter ratio\n";
            cin>>ratio;
            clock_t begin = clock();
            matcher->knnMatch( descriptors1, descriptors2, matchesx, 2 );
            matcher->knnMatch( descriptors2, descriptors1, matchesz, 2 );
            cout << "Matches1-2:" << matchesx.size() << endl;
            cout << "Matches2-1:" << matchesz.size() << endl;
// ratio test proposed by David Lowe paper = 0.8
            std::vector<DMatch> good_matches1, good_matches2;
    // Yes , the code here is redundant, itŝs easy to reconstruct it ....
            for(int i=0; i < matchesx.size(); i++)
             {
               if(matchesx[i][0].distance < ratio * matchesx[i][1].distance)
                   good_matches1.push_back(matchesx[i][0]);
            }
            for(int i=0; i < matchesz.size(); i++)
             {
               if(matchesz[i][0].distance < ratio * matchesz[i][1].distance)
                   good_matches2.push_back(matchesz[i][0]);
            }
            cout << "Good matches1:" << good_matches1.size() << endl;
```

```
        cout << "Good matches2:" << good_matches2.size() << endl;
    // Symmetric Test
        std::vector<DMatch> better_matches;
        for(int i=0; i<good_matches1.size(); i++)
         {
           for(int j=0; j<good_matches2.size(); j++)
         {
              if(good_matches1[i].queryIdx == good_matches2[j].trainIdx && good_matches2[j].queryIdx ==
good_matches1[i].trainIdx)
                 {
                 better_matches.push_back(DMatch(good_matches1[i].queryIdx, good_matches1[i].trainIdx,
good_matches1[i].distance));
                 break;
              }
           }
        }
        cout << "Better matches:" << better_matches.size() << endl;
        clock_t end = clock();
        double elapsed_secs = double(end - begin) / CLOCKS_PER_SEC;
        cout << "Time Costs : " << elapsed_secs << endl;
    // show it on an image
        Mat output;
        drawMatches(image1, kp, image2, kp2, better_matches, output, Scalar::all(-1), Scalar::all(-1),
          vector<char>(), DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS );
        namedWindow("Matches result",WINDOW_KEEPRATIO);
        imshow("Matches result",output);
        waitKey(0);
        destroyAllWindows();
```

```
    break;
}
case 5:
{

    vector<DMatch> matches;
    clock_t begin = clock();

    BFMatcher matcher(NORM_HAMMING,true);
    matcher.match(descriptors1,descriptors2,matches);
    int scene_changed=0;
    vector<DMatch> inliers;
    if(matches.size()>100)
     {
       vector<Point2f> src,dst;
       for(size_t i=0;i<matches.size();i++)
       {
           src.push_back(kp[matches[i].queryIdx].pt);
           dst.push_back(kp2[matches[i].trainIdx].pt);
       }
       vector<uchar> status;
       Mat H=findHomography(srcpoints,dstpoints,status,RANSAC);
       for(size_t i=0;i<matches.size();i++)
      {
           if(status[i]) inliers.push_back(matches[i]);
       }
       if(inliers.size()<80) scene_changed=1;
```

```
        }
        else scene_changed=1;
        if(scene_changed)
         {
           image2.copyTo(image1);
           Ptr<Mat> tmp(new Mat());
           image2.copyTo(*tmp);
           snapshots.push_back(*tmp);
        }
        clock_t end = clock();
        double elapsed_secs = double(end - begin) / CLOCKS_PER_SEC;
        cout << "Time Costs : " << elapsed_secs << endl;
        cout << "matches:" <<inliers.size() << endl;
        Mat matchImage;
        drawMatches(image1,kp,image2,kp2,inliers,matchImage,Scalar::all(-1),Scalar::all(-
1),vector<char>(),DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS);
        namedWindow("Display window",WINDOW_KEEPRATIO);
        imshow("Display window",matchImage);
        waitKey(0);
        destroyAllWindows();
        break;
      }
     default:
        exit(0);
      }
   }
  return 0;
```

```
}
std::vector<DMatch> filter_distance(Mat descriptors,std::vector< DMatch > matches)
{
 double max_dist = 0; double min_dist = 100;
 for( int i = 0; i < descriptors.rows; i++ )
 {
  double dist = matches[i].distance;
  if( dist < min_dist )
    min_dist = dist;
  if( dist > max_dist )
    max_dist = dist;
 }
 printf("-- Max dist : %f \n", max_dist );
 printf("-- Min dist : %f \n", min_dist );
 std::vector< DMatch > good_matches;
 for( int i = 0; i < descriptors.rows; i++ )
 {
  if( matches[i].distance <= max(2*min_dist,0.02) )
   {
    good_matches.push_back( matches[i]); }
   }
 return good_matches;
}
```

APPENDIX – 2 (Tools required to run the application)

· Codeblocks – Version 16.01

Download and installation: http://www.codeblocks.org/downloads/26

· OpenCV – Version 3.0

Download and installation:

- Ubuntu : http://docs.opencv.org/3.0-beta/doc/tutorials/introduction/linux_install/linux_install.html
- Windows : http://docs.opencv.org/3.0-beta/doc/tutorials/introduction/windows_install/windows_install.html

APPENDIX – 3

(How to prepare a computer for demo and how to see the demo)

- This setup can be done on any operating system. But make sure that you install the tools compatible with your operating system.
- Install the tools mentioned in Appendix - 2
- Link all the libraries needed with the Code::blocks IDE
- The complete ORB folder containing orb.cbp and complete dataset with all the images is required.
- In codeblocks click open project and select orb.cbp file
- Build and run the code
- Select the input images.
- Note down the number of keypoints found in the images
- Select the method of matching them
- A window will appear showing the matches between the two images.
- Note down the time and number of matches
- For ratio based matching you need to provide the ratio also.
- For any help, a Readme file is also provided.