COS 470 Image Processing and Computer Vision
2024 Fall
Taylor Brookes
10/15/2024

Assignment 3
Task 1: Feature Detection and Matching (SIFT, ORB)

SIFT:

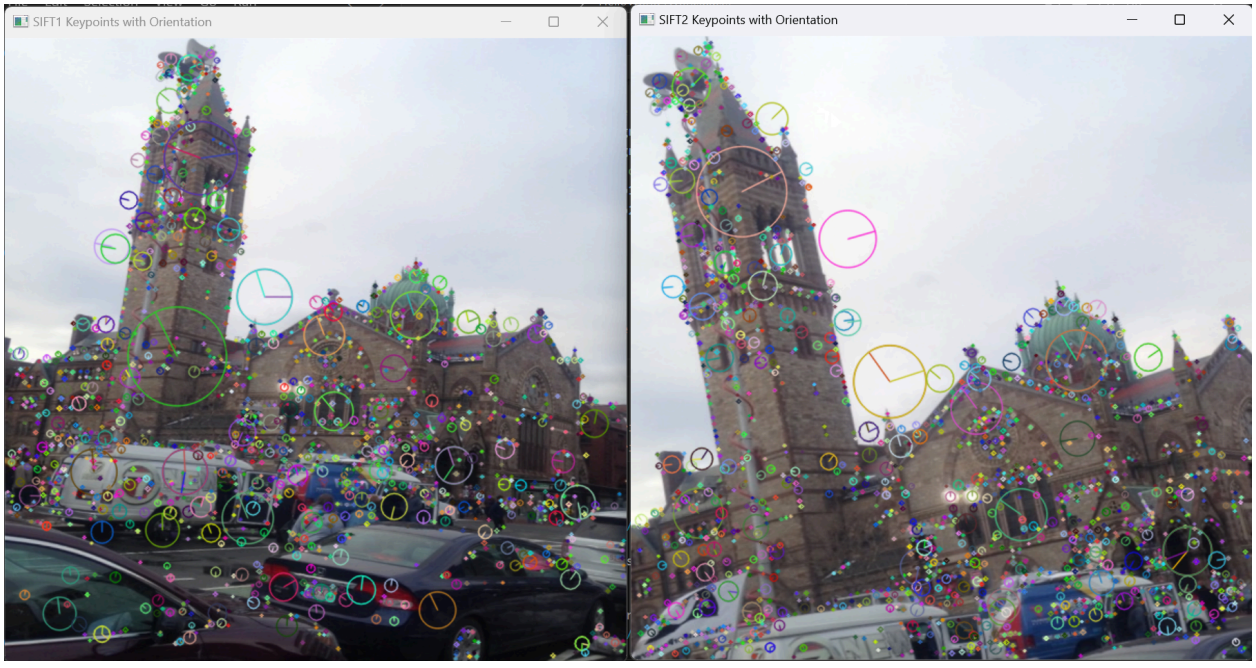
- https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=cc58efc1f17e202a9c196f9df8afd4005d16042a
- https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html
- https://docs.opencv.org/4.x/d7/d60/classcv_1_1SIFT.html

ORB:

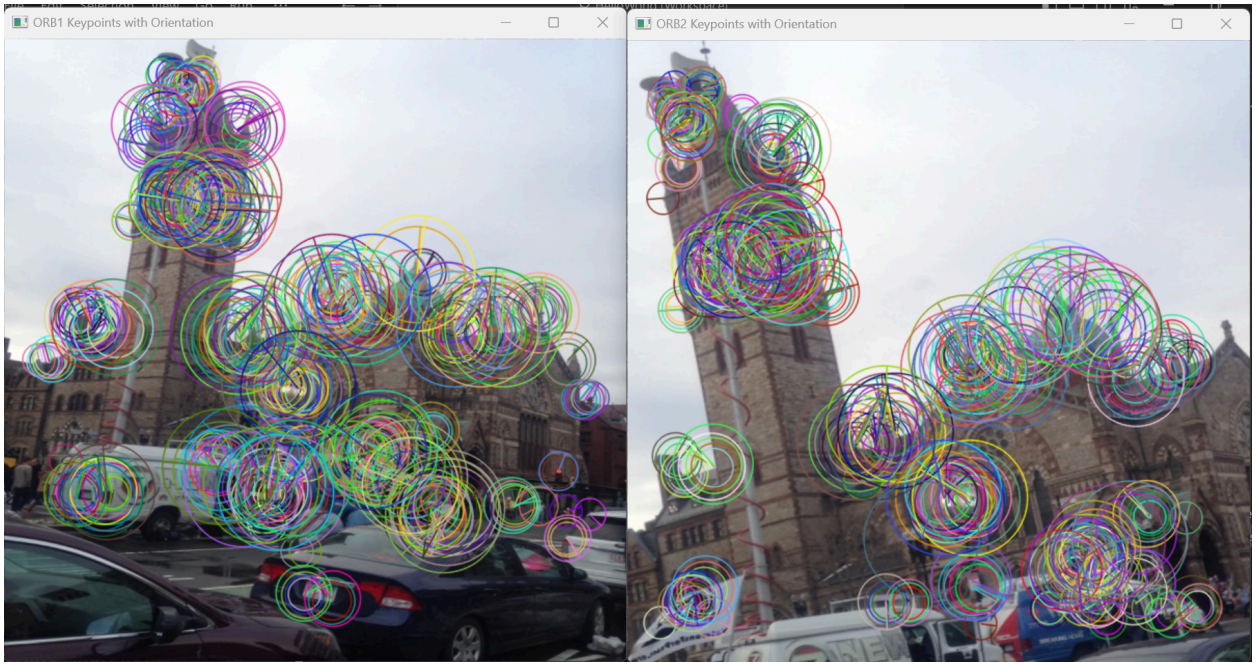
- https://docs.opencv.org/4.x/d1/d89/tutorial_py_orb.html
- https://courses.maine.edu/content/enforced/355524-2510.UMS06-C.0225.1/paper/ORB.pdf

Q1.1: Keypoint Detection -- use SIFT or ORB to detect keypoints in images. Try to increase its accuracy by fine-tuning.
A1.1:

SIFT Keypoints



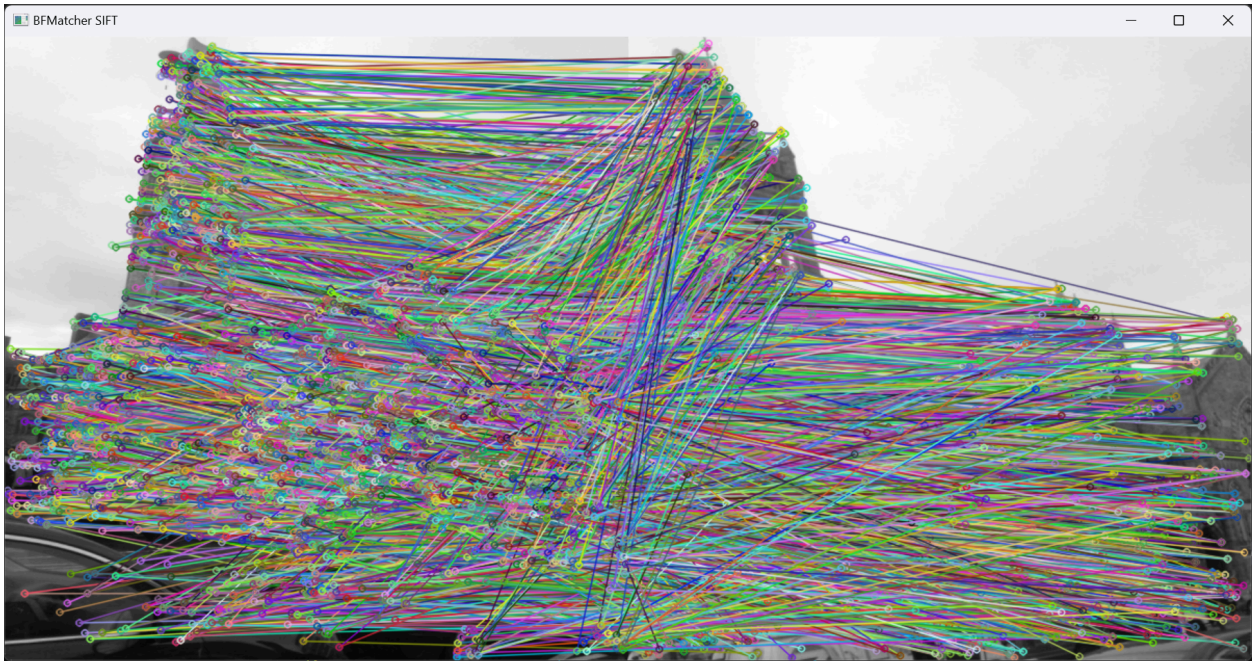
ORB Keypoints



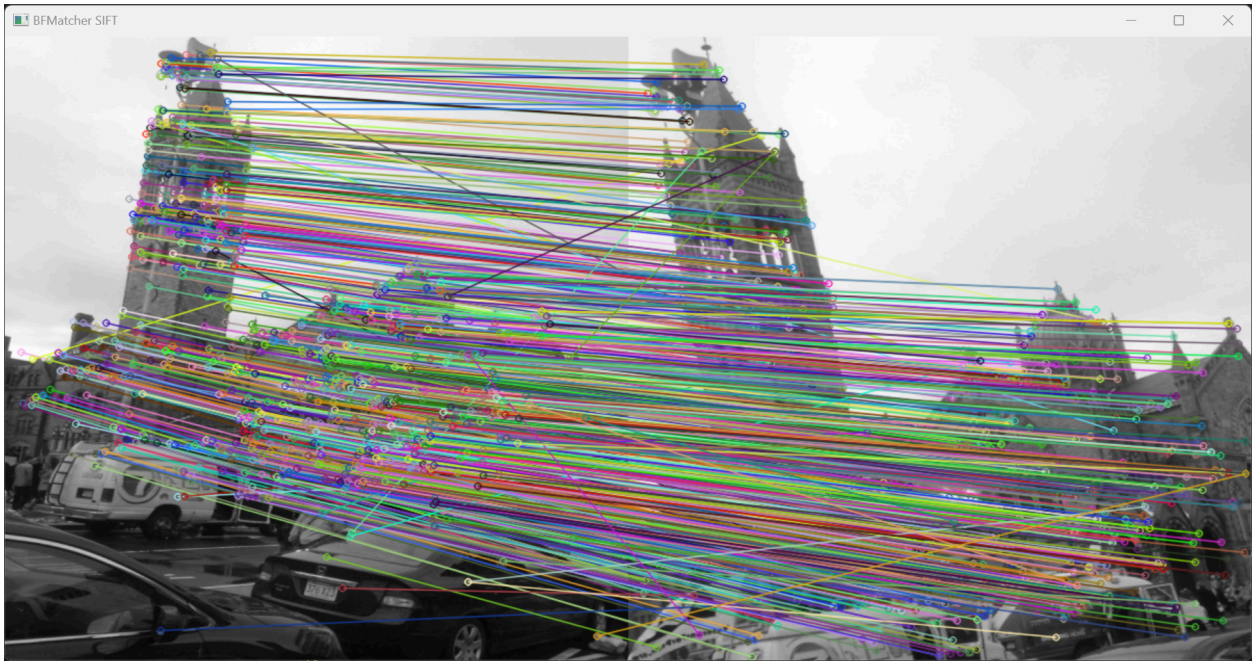
Q1.2 + 1.3: Keypoint Matching -- use FLANN or BFMatcher to match keypoints btw each image pairs. Visualization -- draw lines connecting matched keypoints across image pairs (cv2.drawMatches())

A1.2 + A1.3:

BFMatcher SIFT NNDR 1.0



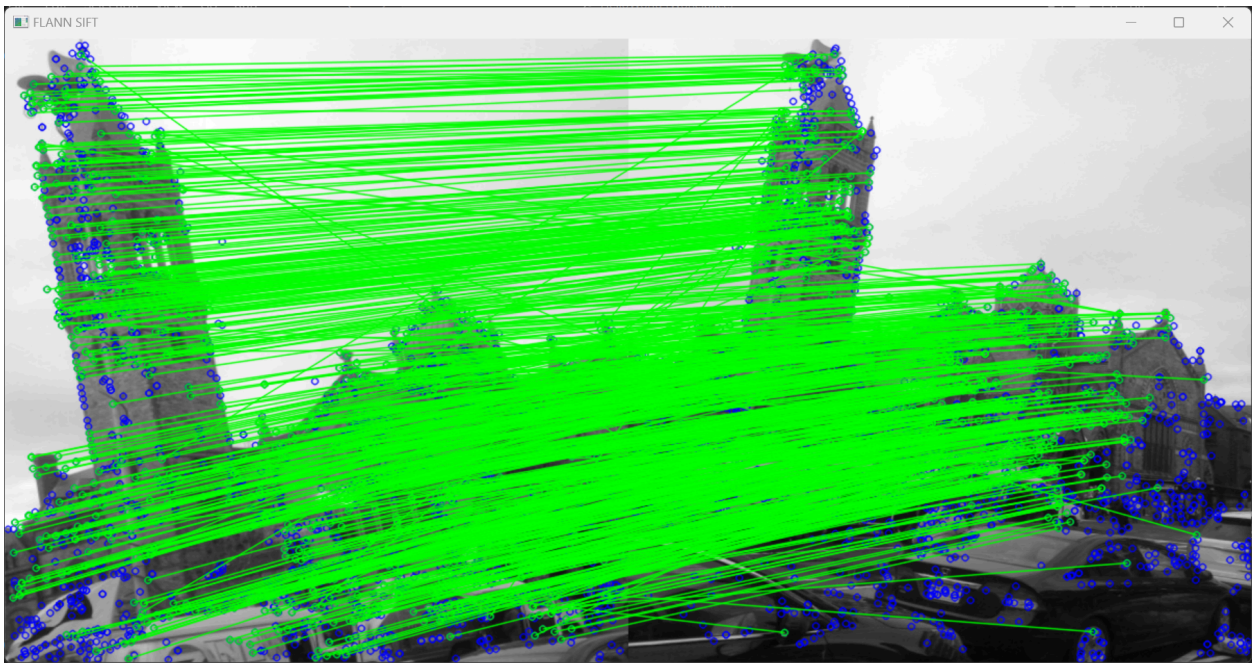
BFMatcher SIFT NNDR 0.8



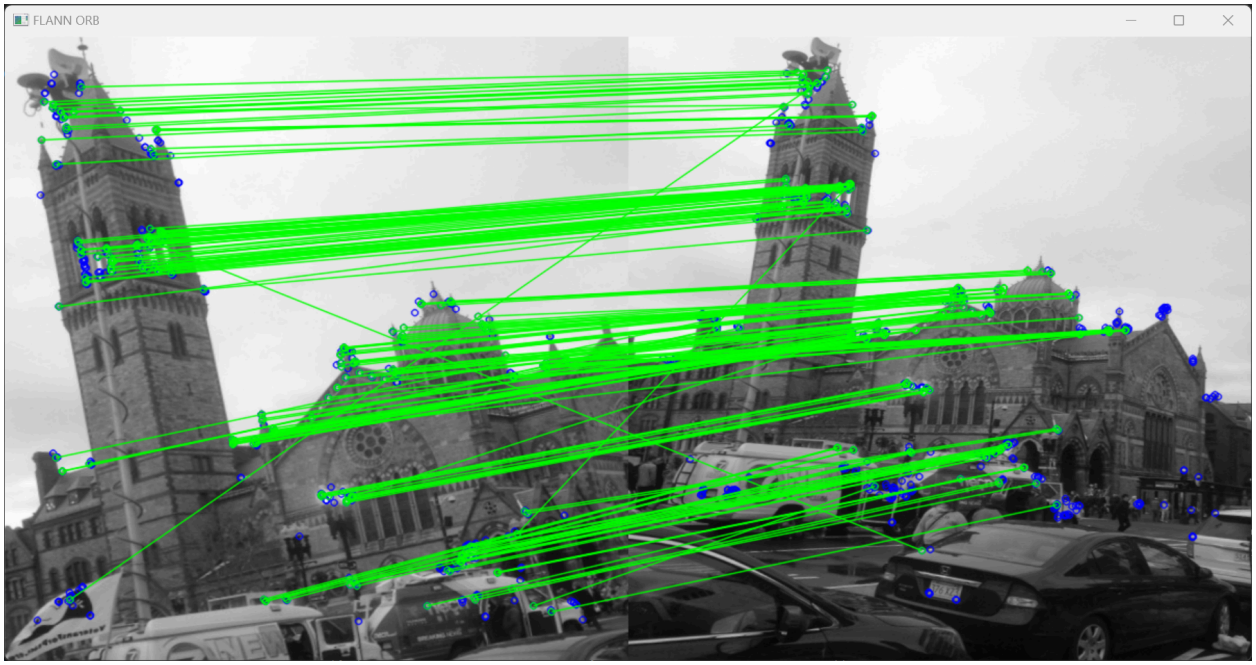
BFMatcher SIFT NNDR 0.8 Tuned

BFMatcher ORB NNDR 0.8

FLANN SIFT NNDR 0.8



FLANN ORB NNDR 0.8



Q1.4: Evaluation of Matches -- report# of good and bad matches, determine accuracy for each image pair. describe how parameter tuning affects the results. Match = good if NNDR < preset threshold (0.8). NNDR compares distance to nearest neighbour with that of the second nearest neighbour, filtering out weaker matches
A1.4:

image/config	matches	good	bad	accuracy
boston SIFT BFMatcher NNDR 0.8	3,148	1,168	1,980	37.10%
boston SIFT FLANN NNDR 0.8	3,148	1,186	1,962	37.67%
boston SIFT BFMatcher NNDR 0.8 swapped	2,280	1,150	1,130	50.44%
boston SIFT FLANN NNDR 0.8 swapped	2,280	1,155	1,125	50.66%
boston ORB BFMatcher NNDR 0.8 500	500	230	270	46.00%
boston ORB FLANN NNDR 0.8 500	500	246	254	49.20%
boston ORB BFMatcher NNDR 0.8 max	9,318	2,776	6,542	29.79%
boston ORB FLANN NNDR 0.8 max	9,318	2,863	6,455	30.73%
boston ORB BFMatcher NNDR 0.8 swapped 500	500	240	260	48.00%

boston ORB FLANN NNDR 0.8 swapped 500	500	243	257	48.60%
boston ORB BFMatcher NNDR 0.8 swapped max	5,427	2,581	2,846	47.56%
boston ORB FLANN NNDR 0.8 swapped max	5,427	2,615	2,812	48.19%
boston SIFT BFMatcher NNDR 0.8 TUNED	377	235	142	62.33%
boston SIFT FLANN NNDR 0.8 TUNED	377	235	142	62.33%
boston ORB BFMatcher NNDR 0.8 TUNED	500	258	242	51.60%
boston ORB FLANN NNDR 0.8 TUNED	500	261	239	52.20%
castle SIFT BFMatcher NNDR 0.8	34,718	10,469	24,249	30.15%
castle SIFT FLANN NNDR 0.8 swapped	29,480	10,274	19,206	34.85%
castle ORB BFMatcher NNDR 0.8 500	500	269	231	53.80%
castle ORB FLANN NNDR 0.8 max	127,075	51,792	75,283	40.76%
castle SIFT BFMatcher NNDR 0.8 TUNED	3,206	2,059	1,147	64.22%
rushmore SIFT BFMatcher NNDR 0.8	17,169	1,180	15,989	6.87%
rushmore SIFT FLANN NNDR 0.8 swapped	30,444	1,640	28,804	5.39%
rushmore ORB BFMatcher NNDR 0.8 500	500	30	470	6.00%
rushmore ORB FLANN NNDR 0.8 max	48,599	4,054	44,545	8.34%
rushmore SIFT BFMatcher NNDR 0.8 TUNED	4,337	876	2,461	20.20%
dame SIFT BFMatcher NNDR 0.8	24,548	2,557	21,991	10.42%
dame SIFT FLANN NNDR 0.8 swapped	20,376	2,582	17,794	12.67%
dame ORB BFMatcher NNDR 0.8 500	500	19	481	3.80%
dame ORB FLANN NNDR 0.8 max	86,458	8,946	77,512	10.35%
dame SIFT BFMatcher NNDR 0.8 sigma=5.0	2,613	698	1,915	26.71%

Tuning:

category	total	good	bad	percent
boston ORB BFMatcher original NNDR 0.8	500	230	270	46.00%
swapped	500	240	260	48.00%

swapped nfeatures=max (default 500)	5,427	2,581	2,846	47.56%
swapped scaleFactor=2 (default 1.2)	441	194	247	43.99%
swapped scaleFactor=1.05	500	125	375	25.00%
swapped nlevels=4 (default 8)	500	213	287	42.60%
swapped nlevels=16	454	219	235	48.25%
swapped nlevels=32	432	216	216	50.00%
swapped edgeThreshold=15 (default 31)	500	258	242	51.60%
swapped edgeThreshold=63	466	190	276	40.77%
swapped firstLevel=1 (default 0)	500	243	257	48.60%
swapped firstLevel=2 (default 0)	500	240	260	48.00%
swapped WPA_K=3, cv2.NORM_HAMMING2 (default 2, HAMMING	500	226	274	45.20%
swapped WPA_K=4, HAMMING2	500	223	277	44.60%
swapped cv2.ORB_FAST_SCORE (default cv2.ORB_HARRIS_SCORE)	507	240	267	47.34%
swapped patchSize=15 (default 31)	500	157	343	31.40%
swapped patchSize=63	500	243	257	48.60%
swapped edgeThreshold=15, patchSize=63	500	239	261	47.80%
swapped fastThreshold=10 (default 20)	500	240	260	48.00%
swapped fastThreshold=40	495	239	256	48.28%

category	total	good	bad	percent
boston ORB FLANN original NNDR 0.8	500	243	257	48.60%
… mostly the same, all below is also swapped				
swapped edgeThreshold=15 (default 31)	500	261	239	52.20%
index_params table_number=12	500	242	258	48.40%
index_params key_size=20	500	186	314	37.20%

index_params multi_probe_level=2	500	241	259	48.20%
index_params ^ 3 above combined	500	256	244	51.20%
search_params checks=25 (default 50?)	500	243	257	48.60%
search_params checks=100	500	243	257	48.60%

category	total	good	bad	percent
boston SIFT BFMatcher original NNDR 0.8	3,148	1,168	1,980	37.10%
swapped	2,280	1,150	1,130	50.44%
swapped nfeatures=500 (default 0)	500	241	259	48.20%
swapped nfeatures=100,000 (same as 0)	2,280	1,150	1,130	50.44%
swapped nOctaveLayers=2 (default 3)	1,934	1,032	902	53.36%
swapped nOctaveLayers=4	2,498	1,286	1,212	51.48%
swapped contrastThreshold=0.02 (default 0.04)	4,029	2,012	2,017	49.94%
swapped contrastThreshold=0.06	1,220	604	616	49.51%
swapped edgeThreshold=5 (default 10)	1,750	886	864	50.63%
swapped edgeThreshold=20	2,554	1,295	1,259	50.70%
swapped sigma=1.2 (default 1.6)	3,034	1,347	1,687	44.40%
swapped sigma=2.0	1,731	931	800	53.78%
swapped enable_precise_upscale=True (default false)	2,245	1,101	1,144	49.04%
swapped nOctaveLayers=2, edgeThreshold=5	1,501	803	698	53.50%
swapped nOctaveLayers=2, edgeThreshold=20	2,186	1,159	1,027	53.02%
swapped nOctaveLayers=2, sigma=2.0	1,387	758	629	54.65%
swapped edgeThreshold=5, sigma=2.0	1,319	710	609	53.83%
swapped edgeThreshold=20, sigma=2.0	1,957	1,060	897	54.16%
swapped sigma=3.0	841	512	329	60.88%
swapped sigma=5.0	377	235	142	62.33%

category	total	good	bad	percent
boston SIFT FLANN original NNDR 0.8	3,148	1,186	1,962	37.67%
swapped	2,280	1,155	1,125	50.66%
… all almost the same, all below is also swapped				
swapped sigma=5.0	377	235	142	62.33%
index_params trees=2 (default 5)	2,280	1,156	1,124	50.70%
index_params trees=10	2,280	1,152	1,128	50.53%
search_params checks=25 (default 50?)	2,280	1,161	1,119	50.92%
search_params checks=100	2,280	1.152	1,128	50.53%

I was getting an error on FLANN ORB: “ValueError: not enough values to unpack (expected 2, got 1)”, so I googled it and found out that sometimes there aren't matches for the pairs, so I implemented the try-catch block solution found here: https://stackoverflow.com/questions/55612455/when-receiving-valueerror-not-enough-values-to-unpack-expected-2-got-1-ho

NOTES:
max → nfeatures=0
500 → parameter not added, default value is 500
swapped → img1 (query) = Boston1.jpeg, img2 (train) = Boston.jpeg
TUNED → messed with each parameter one by one to get as high as possible

Parameter tuning refines the criteria for matches, EX: “swapped sigma=2.0” means the images have been processed in reverse order (EX: Boston1 as query and Boston as training), and that the Gaussian Filter’s sigma has been changed from the default 1.6 to 2.0. Sigma in a Gaussian Filter changes the standard deviation of the filter, which in turn defines the amount of blurring that takes place in the preprocessing of the image. A larger sigma means the image is blurred more.

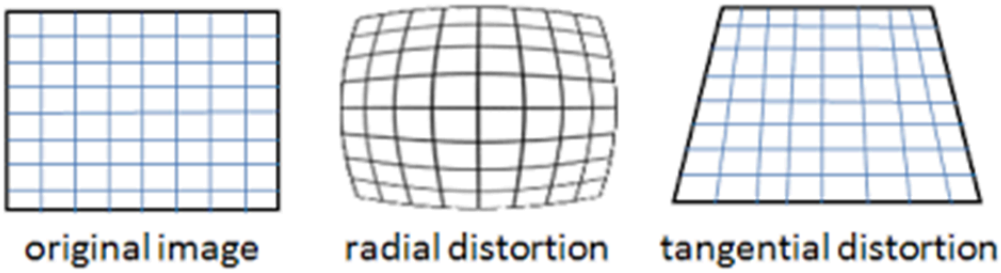
Taks2: Application of Feature Matching for Depth Estimation
Q2: explain the concepts, methods, theory, and step-by-step of how feature matching can be used for depth information and estimation from images.
A2:

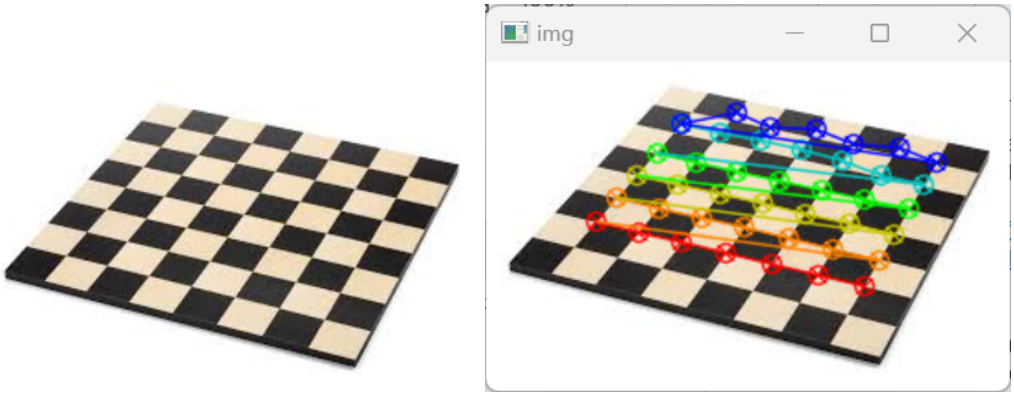
Camera Calibration

Cameras create distortion, with the two main kinds being Radial Distortion and Tangential Distortion. Radial distortion curves an image around the corners of the picture, while Tangential distortion has closer objects appear larger than they actually are.

There are intrinsic and extrinsic parameters of the camera. Intrinsic parameters include focal length and optical centers, while extrinsic parameters include rotation and translation vectors which occur due to 3D space being compressed into 2D.

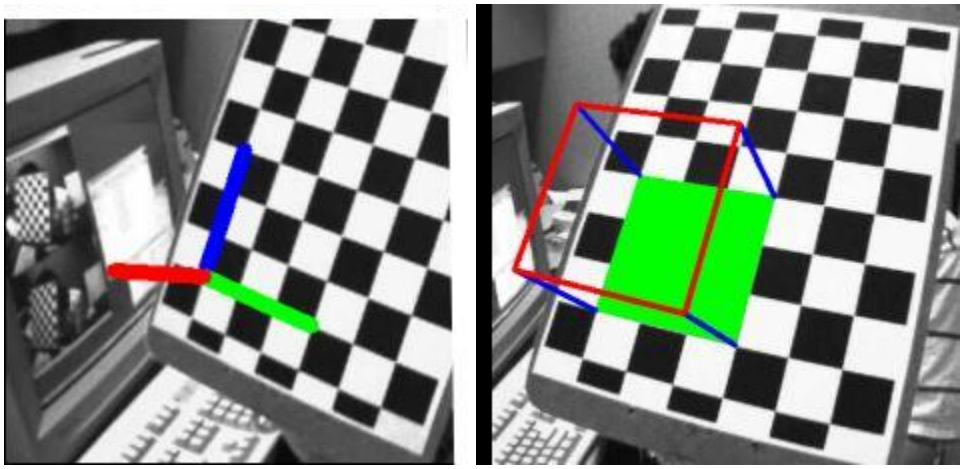
The important input data needed for calibration of the camera is the set of 3D real world points and the corresponding 2D coordinates of these points in the image. These require the X,Y,Z coordinates of a set of images to calibrate the camera. For simplicity, the article keeps the chess board on a constant Z axis and while moving the camera along the Z axis, which only requires knowing the values of X,Y now. To calibrate the camera, we find the corners of the images in the set being used, in this case of chessboards, with cv2.findChessboardCorners(). 3D points are object points and 2D points are image points. We first create the calibration for the camera with cv2.calibrateCamera() with the object and image points as inputs. For a new input image that we would like to undistort, we use cv2.getOptimalNewCameraMatrix() to create a matrix based on a free scaling parameter, and it also creates a ROI (region of interest) for the input image. We then use cv2.undistort() and crop the undistorted image by the ROI. To find how close the new undistorted image is to what it should be, we use cv2.projectPoints(), calculate error with cv2.norm, and then find the mean. A high error score is worse than a low error score.





Pose Estimation

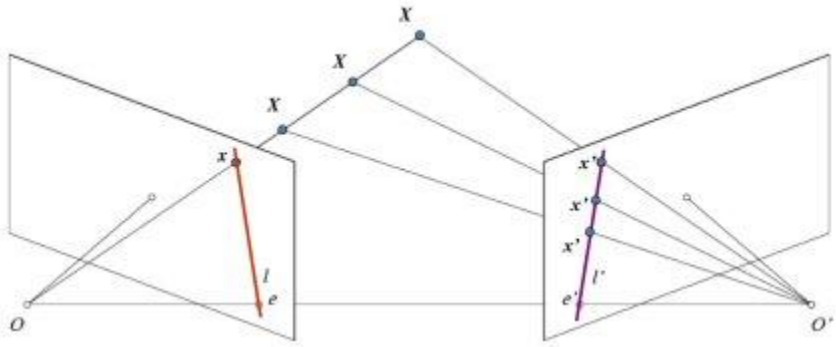
We now can calculate how a new image taken from the same camera is oriented in space. To do this, we create lines in space for the representation of each axis (X,Y,Z) using cv2.line(). We then find the interior grid of the chessboard input image, use cv2.solvePnPRansac() to calculate rotation and translation, and draw the projection of the axis points with a 21 line generateImage() function. These can either be lines in 3D space or a cube in 3D space.



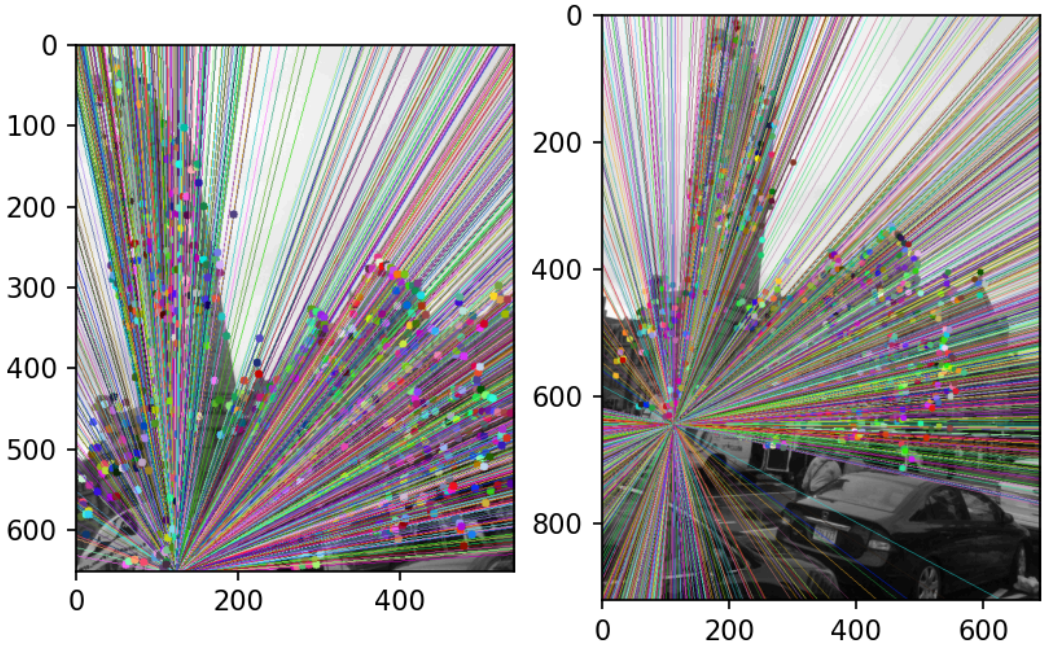
Epipolar Geometry

Depth information is able to be found through the use of more than one image taken of one subject at a different perspective. Stereo vision is the use of 2 images, and epipolar geometry is the geometry of stereo vision. We use the two images to triangulate 2D image points into 3D object points. The projection of the X point from the line OX onto the right plane forms line L' with points X' along line OX, which all the X' points from the right plane translate to the single X point on the left plane. Line L' is called an epiline. The epipolar constraint is the location of X along line L', which can be any of the values of X' until the right one is found in the 3D space. The epipolar plane is the plane between O, X, and O', which spans 180deg of O and 180deg of O' along the respective epilines. E is the point on the epiline that is the location of O' (the camera center of the right plane), called an epipole, which may sometimes be located outside the image, meaning one camera may not be able to see the other. The location of the

epipole is found through the use of many epilines and the location of their intersection points. To find these, we need the Essential Matrix (translation and rotation) and the Fundamental Matrix (the essential matrix with the addition of information about the intrinsics of both cameras used to relate the two cameras in pixel coordinates, EX: focal length and optical centers). The fundamental matrix maps a point from one image (plane) to the epiline of another image (plane), and we need at least 8 of these points, calculated by matching points from both images. To find these matches, we use SIFT FLANN with the ratio test to find good matches between the two images, then find the fundamental matrix with cv2.findFundamentalMatrix(), select only inlier points, then find the epilines and draw them on both images. The point at which the epilines converge is the epipole of the other image.



planes, epilines, epipoles



epiline generation from the provided code

Depth Map from Stereo Images

The depth of a point in a scene is inversely proportional to the difference in distance of corresponding image points (2D) and their camera centers (epipoles). This can be used to find the depth of all pixels in an image. When matches are found, we calculate the disparity (disparity map). Noise can be created in this new map, and reduced by adjusting parameters with getters and setters.

Disparity = x - x' = Bf / Z

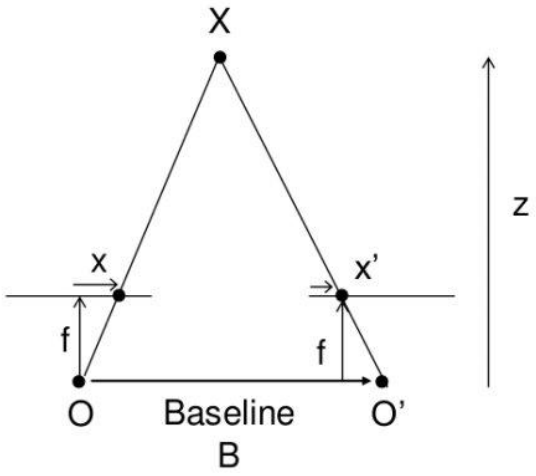
x = distance between point in image plane corresponding to O

x' = same, corresponding to O'

B = distance between the two camera (known, epipoles)

f = focal length of camera (known)

Z = length from O / O' to X along one axis



How Feature Mapping is used for Depth Estimation

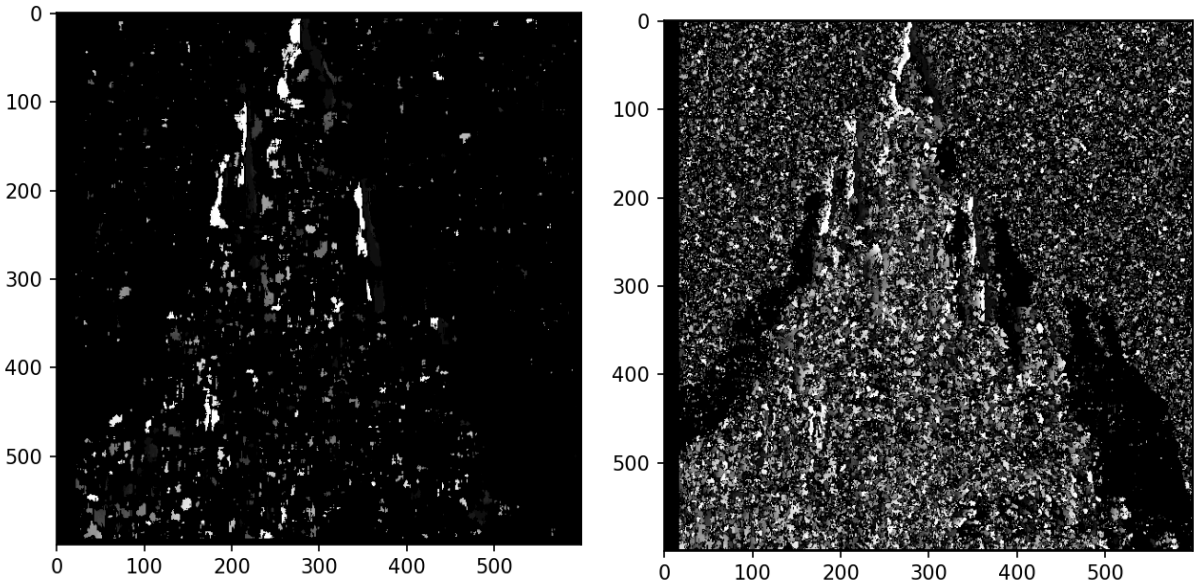
Feature mapping can be used for depth estimation by creating a depth map from a set of stereo images. The quality of the feature map is dependent upon the depths in the images and the amount of noise present in the images.

1. read 2 images in: a left and a right image of the same subject but at different perspectives
2. create a stereo object
3. calculate a disparity map of the image

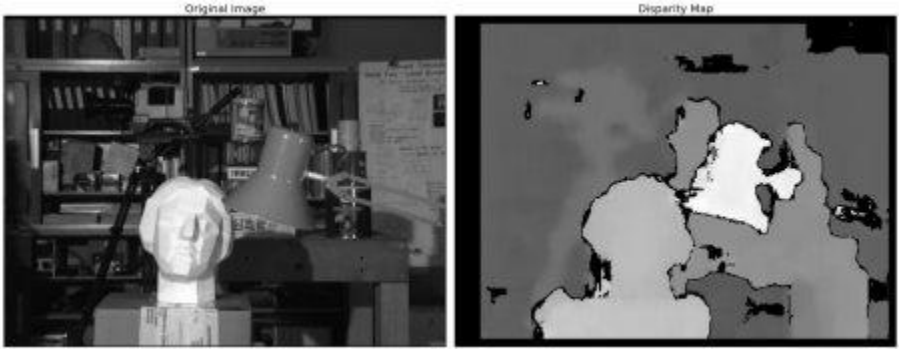
```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt

imgL = cv.imread('Boston1.jpeg', cv.IMREAD_GRAYSCALE)
imgL = cv.resize(imgL, (600,600))
imgR = cv.imread('Boston.jpeg', cv.IMREAD_GRAYSCALE)
imgR = cv.resize(imgR, (600,600))
```

```
stereo = cv.StereoBM.create(numDisparities=16, blockSize=15)
disparity = stereo.compute(imgL,imgR)
plt.imshow(disparity,'gray')
plt.show()
```



The above image is a disparity map between Castle.jpg and Castle1.jpg. The outline of the castle is somewhat visible, with different shades indicating different depths. Left is blockSize=15, right is blockSize = 5.



the provided disparity map from the article