HOMEWORK 2A
ECE/CS 8690 2302 Computer Vision

Feature Detection, Description, and Matching

Xuanbo Miao

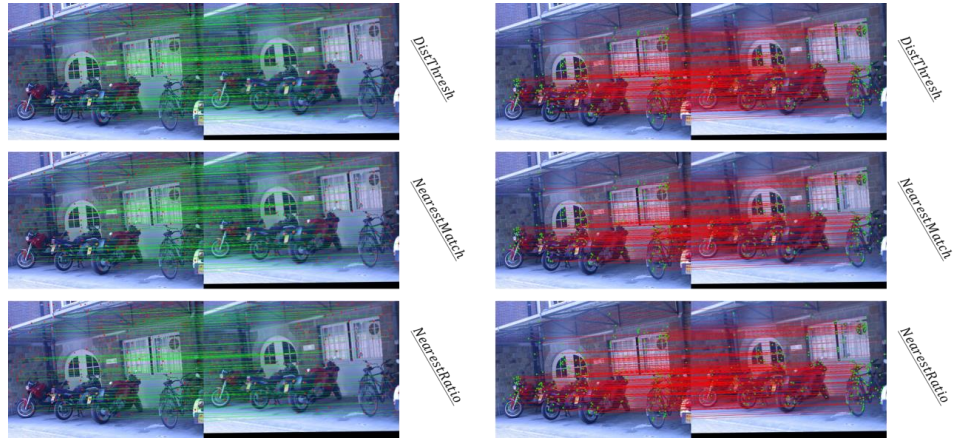
14422044

xmiao@mail.missouri.edu

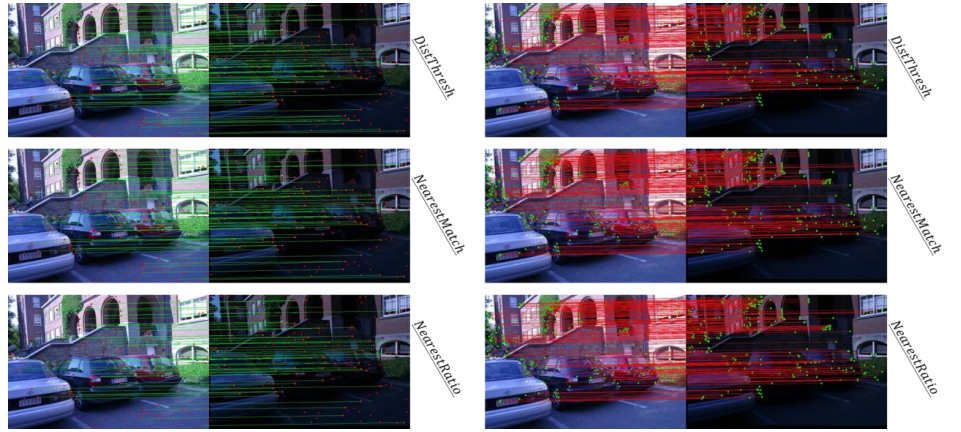
Abstract

This assignment focuses on implementing a basic pipeline for feature detection, description, and matching. The first step is feature detection, which involves computing the autocorrelation matrix and a scalar feature detection measure using a formula discussed in class. The detected feature points are then shown in the image and included in the report. The second step is feature matching and evaluation, which involves defining and describing a feature descriptor and using it to match features using three different strategies. The performance of the feature matching is evaluated using transformation/homography matrices provided with the data.

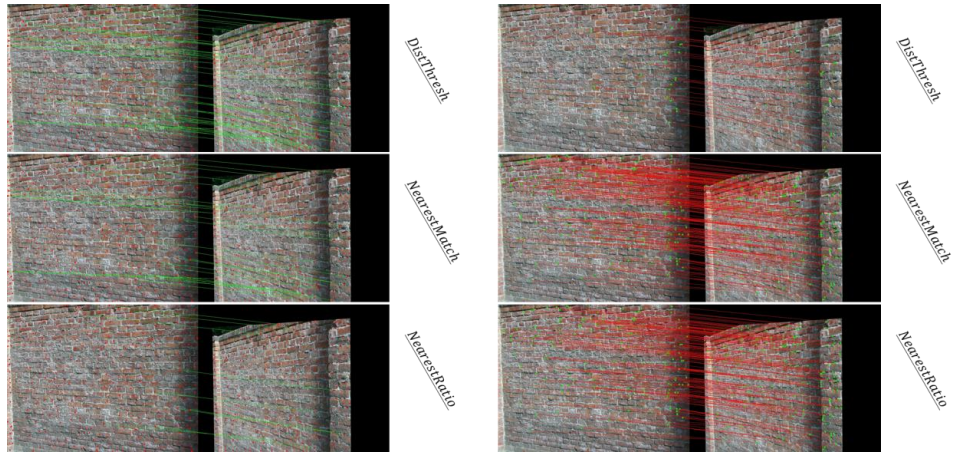
Bikes
Img 1 – 3
①



Leuven
Img 1 – 6
②



Wall
Img 2 – 4
③



Functions as required

1. function [R]=computeFeatureMeasure(I,parameters)

```python
def computeFeatureMeasure(I, k=0.04, window_size=3):

    auto_rr_matrix = auto_correlation_matrix(I, window_size)
    [xx, yy, xy] = auto_rr_matrix
    xx = gaussian_smooth(xx)
    yy = gaussian_smooth(yy)
    xy = gaussian_smooth(xy)
    # Compute the Harris response
    det = xx * yy - xy ** 2
    trace = xx + yy
    R = (det+1)/ (trace+1)
    R /= np.mean(R)
    #print(np.max(abs(R)))
    return R

```

2. function [x,y,mask]=FeatureMeasure2Points(R,npoints)

```python
def FeatureMeasure2Points(R, npoints,threshold = 0.01):

    # Find the locations of local maxima above the threshold
    h, w = R.shape
    mask = np.zeros((h, w), np.uint8)
    mask[R > threshold] = 1

    # Use morphological dilation to find the connected regions
    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
    mask_erosion1 = cv2.erode(mask,          kernel,iterations = 2)
    mask_dilate1 = cv2.dilate(mask_erosion1, kernel,iterations = 2)
    mask_erosion2 = cv2.erode(mask_dilate1,  kernel,iterations = 5)
    mask_dilate2 = cv2.dilate(mask_erosion2, kernel,iterations = 2)
    mask_erosion3 = cv2.erode(mask_dilate2,  kernel,iterations = 5)
    mask_dilate3 = cv2.dilate(mask_erosion3, kernel,iterations = 2)
    mask_erosion4 = cv2.erode(mask_dilate3,  kernel,iterations = 5)
    mask_dilate4 = cv2.dilate(mask_erosion4, kernel,iterations = 2)

    #cv2.imshow('imgh',mask_dilate1*250)
    # Compute the centroid of each connected region as the feature point location
    num_labels, labels, stats1, centroids1 = cv2.connectedComponentsWithStats(mask_dilate1)
    num_labels, labels, stats2, centroids2 = cv2.connectedComponentsWithStats(mask_dilate2)
    num_labels, labels, stats3, centroids3 = cv2.connectedComponentsWithStats(mask_dilate3)
    num_labels, labels, stats4, centroids4 = cv2.connectedComponentsWithStats(mask_dilate4)

    # Remove the background label

    centroids_=np.concatenate((centroids1[1:],centroids2[1:],centroids3[1:],centroids4[1:]), axis=0)
    stats_=np.concatenate((stats1[1:],stats2[1:],stats3[1:],stats4[1:]), axis=0)
    stats_=stats_[:,4]
    # Sample a fixed number of feature points from the list of all feature points
    centroids=[]
    for idx in range(len(stats_)):
        if 100 < stats_[idx] < 500:
            centroids=np.concatenate((centroids, centroids_[idx]), axis=0)
    centroids=np.reshape(centroids, (int(len(centroids)/2),2))

    if len(centroids) > npoints:
        centroids = centroids[:npoints]

    # Return the x, y coordinates of the feature points and the mask of the detected regions
    x = centroids[:, 0]
    y = centroids[:, 1]
    mask = np.zeros_like(R)
    for point in centroids:
        mask[int(point[1]), int(point[0])] = 1

    return x, y, mask
```

3. function [DList]=generateFeatureDescriptors(I,x,y)

```python
def generateFeatureDescriptors(I, x, y, patch_size=21):
    # Define a square patch around each feature point
    half_patch_size = patch_size // 2
    num_points = len(x)
    Dlist = np.zeros((num_points, patch_size ** 2), dtype=np.float32)
    for i in range(num_points):
        patch=extract_bordered_patch(I, x[i], y[i], patch_size,patch_size , border=patch_size)
        norm_patch = (patch - np.mean(patch)) / np.std(patch)
        norm_kernel = np.ones(norm_patch.shape) / norm_patch.size
        inc_patch = cv2.filter2D(norm_patch, -1, norm_kernel)
        inc_descriptor=inc_patch.flatten()
        inc_descriptor /= np.linalg.norm(inc_descriptor)
        Dlist[i, :] = inc_descriptor

    return Dlist
```

4. function [Dist]=computeDescriptorDistances(Dlist1,Dlist2)

```python
def evaluate_INC(descriptor1, descriptor2):

    eps = 1e-6 # A small number to prevent division by zero
    mag1 = np.linalg.norm(descriptor1) + eps
    mag2 = np.linalg.norm(descriptor2) + eps
    dot_product = np.dot(descriptor1, descriptor2)
    inc = 1.0 - dot_product / (mag1 * mag2)
    if dot_product>1:
        dot_product=dot_product
    return inc

def computeDescriptorDistances(Dlist1, Dlist2):
    Dist = np.zeros((Dlist1.shape[0], Dlist2.shape[0]))
    for i in range(Dlist1.shape[0]):
        for j in range(Dlist2.shape[0]):
            #dist[i, j] = np.sqrt(np.sum((Dlist1[i] - Dlist2[j])**2))
            Dist[i, j] = evaluate_INC(Dlist1[i], Dlist2[j])

    return Dist
```

5. Function [MatchList1]=Distance2Matches_DistThresh(Dist,Th1)
   Function [MatchList2]=Distance2Matches_NearestMatch(Dist,Th2)
   Function [MatchList3]=Distance2Matches_NearestRatio(Dist,Th3

```python
def Distance2Matches_DistThresh(Dist, Th1=100):
    # Distance threshhold
    # Apply distance thresholding and select good matches
    MatchList1 = []
    for i in range(Dist.shape[0]):
        for j in range(Dist.shape[1]):
            if Dist[i, j] < Th1:
                MatchList1.append([i, j])

    return MatchList1


def Distance2Matches_NearestMatch(Dist, Th2=0.6):
    # Nearest match
    # Sort matches by distance and select top x
    Dlist = []
    for i in range(Dist.shape[0]):
        for j in range(Dist.shape[1]):
            Dlist.append([Dist[i, j], i, j])
    Dlist = sorted(Dlist, key=lambda x: x[0])
    MatchList2_ = Dlist[:int(len(Dlist)*Th2/len(Dist))]

    MatchList2 = []
    for m in MatchList2_:
        MatchList2.append(m[1:3])

    return MatchList2


def Distance2Matches_NearestRatio(Dist, Th3=0.7):
    # NNDR

    DD = []
    for i in range(Dist.shape[0]):
        Dlist = []
        for j in range(Dist.shape[1]):
            Dlist.append([Dist[i, j], i, j])
        Dlist = sorted(Dlist, key=lambda x: x[0])
        DD.append(Dlist[:2])

    MatchList3 = []
    for DDD in DD:
        if DDD[0][0] < Th3 * DDD[1][0]:
            MatchList3.append(DDD[0][1:3])

    return MatchList3
```

All Coded Functions

```
26 > def read_ppm_file(file_path): ···
47
48 > def unify_contrast_brightness(image): ···
65
66 > def unify_image(img, clip_limit=2.0, tile_size=8, brightness=128, contrast=1.0, detail_level=1.0): ···
88
89 > def auto_correlation_matrix(image, window_size=3): ···
103
104 > def gaussian_smooth(image, sigma=2): ···
119
120 > def computeFeatureMeasure(I, k=0.04, window_size=3): ···
134
135 > def FeatureMeasure2Points(R, npoints,threshold = 0.01): ···
182
183 > def extract_bordered_patch(img, x, y, w, h, border): ···
202
203 > def generateFeatureDescriptors(I, x, y, patch_size=21): ···
218
219 > def show_descriptor_image(image, x, y, Dlist, patch_size=21): ···
231
232 > def transform_point(H1to2, x, y): ···
239
240 > def eval_TFPN_s(kp1, kp2, selected_matches, all_matches, trans_matrix, judge_threshhold=16): ···
283
284 > def eval_TFPN(kp1_x, kp1_y, kp2_x, kp2_y,Dist,Matchlist, trans_matrix, judge_threshhold=16): ···
322
323 > def print_Rarray(arrayname, number_array): ···
333
334 > def DetectAndCompute(image, max_points): ···
345
346 > def evaluate_INC(descriptor1, descriptor2): ···
356
357 > def computeDescriptorDistances(Dlist1, Dlist2): ···
365
366 > def computeDescriptorMatchs_s(Dlist1, Dlist2): ···
371
372 > def draw_correspondences(image1, image2, points,color1,color2): ···
391
392 > def post_Distance2Matches_s(MatchList, All_MatchList, kp1, kp2, img_o, img_t, trans_matrix): ···
399
400 > def Distance2Matches_DistThresh(Dist, Th1=100): ···
410
411 > def Distance2Matches_NearestMatch(Dist, Th2=0.6): ···
426
427 > def Distance2Matches_NearestRatio(Dist, Th3=0.7): ···
444
445 > def Distance2Matches_DistThresh_s(AllMatchList, threshold_match_d=100): ···
454
455 > def Distance2Matches_NearestMatch_s(AllMatchList, threshold_match_n=0.6): ···
465
466 > def Distance2Matches_NearestRatio_s(AllMatchList, threshold_match_nn=0.7): ···
```

Output dialog of ‘bikes’ img *1* –img *3*

```
22:14:06
./bikes/H2p_
FixDist: P: 100, N: 43342, TP: 83, FN: 17, FP: 1536, TN: 41806, TPR: 0.830, FPR: 0.035, PPV: 0.051, ACC: 0.964,
Nearest: P: 100, N: 43342, TP: 85, FN: 15, FP: 1985, TN: 41357, TPR: 0.850, FPR: 0.046, PPV: 0.041, ACC: 0.954,
N N D R: P: 100, N: 43342, TP: 69, FN: 31, FP: 98, TN: 43244, TPR: 0.690, FPR: 0.002, PPV: 0.413, ACC: 0.997,

FixDist_s: P: 390, N: 110, TP: 356, FN: 34, FP: 4, TN: 106, TPR: 0.913, FPR: 0.036, PPV: 0.989, ACC: 0.924,
Nearest_s: P: 390, N: 110, TP: 297, FN: 93, FP: 3, TN: 107, TPR: 0.762, FPR: 0.027, PPV: 0.990, ACC: 0.808,
N N D R_s: P: 390, N: 110, TP: 374, FN: 16, FP: 3, TN: 107, TPR: 0.959, FPR: 0.027, PPV: 0.992, ACC: 0.962,

./bikes/H3p_
FixDist: P: 95, N: 40779, TP: 81, FN: 14, FP: 1852, TN: 38927, TPR: 0.853, FPR: 0.045, PPV: 0.042, ACC: 0.954,
Nearest: P: 95, N: 40779, TP: 83, FN: 12, FP: 1865, TN: 38914, TPR: 0.874, FPR: 0.046, PPV: 0.043, ACC: 0.954,
N N D R: P: 95, N: 40779, TP: 64, FN: 31, FP: 101, TN: 40678, TPR: 0.674, FPR: 0.002, PPV: 0.388, ACC: 0.997,

FixDist_s: P: 396, N: 104, TP: 364, FN: 32, FP: 6, TN: 98, TPR: 0.919, FPR: 0.058, PPV: 0.984, ACC: 0.924,
Nearest_s: P: 396, N: 104, TP: 297, FN: 99, FP: 3, TN: 101, TPR: 0.750, FPR: 0.029, PPV: 0.990, ACC: 0.796,
N N D R_s: P: 396, N: 104, TP: 385, FN: 11, FP: 3, TN: 101, TPR: 0.972, FPR: 0.029, PPV: 0.992, ACC: 0.972,

./bikes/H4p_
FixDist: P: 95, N: 42919, TP: 78, FN: 17, FP: 2133, TN: 40786, TPR: 0.821, FPR: 0.050, PPV: 0.035, ACC: 0.950,
Nearest: P: 95, N: 42919, TP: 78, FN: 17, FP: 1972, TN: 40947, TPR: 0.821, FPR: 0.046, PPV: 0.038, ACC: 0.954,
N N D R: P: 95, N: 42919, TP: 64, FN: 31, FP: 99, TN: 42820, TPR: 0.674, FPR: 0.002, PPV: 0.393, ACC: 0.997,

FixDist_s: P: 389, N: 111, TP: 358, FN: 31, FP: 7, TN: 104, TPR: 0.920, FPR: 0.063, PPV: 0.981, ACC: 0.924,
Nearest_s: P: 389, N: 111, TP: 296, FN: 93, FP: 4, TN: 107, TPR: 0.761, FPR: 0.036, PPV: 0.987, ACC: 0.806,
N N D R_s: P: 389, N: 111, TP: 379, FN: 10, FP: 1, TN: 110, TPR: 0.974, FPR: 0.009, PPV: 0.997, ACC: 0.978,

./bikes/H5p_
FixDist: P: 99, N: 42487, TP: 78, FN: 21, FP: 1901, TN: 40586, TPR: 0.788, FPR: 0.045, PPV: 0.039, ACC: 0.955,
Nearest: P: 99, N: 42487, TP: 78, FN: 21, FP: 1951, TN: 40536, TPR: 0.788, FPR: 0.046, PPV: 0.038, ACC: 0.954,
N N D R: P: 99, N: 42487, TP: 61, FN: 38, FP: 102, TN: 42385, TPR: 0.616, FPR: 0.002, PPV: 0.374, ACC: 0.997,

FixDist_s: P: 398, N: 102, TP: 365, FN: 33, FP: 4, TN: 98, TPR: 0.917, FPR: 0.039, PPV: 0.989, ACC: 0.926,
Nearest_s: P: 398, N: 102, TP: 298, FN: 100, FP: 2, TN: 100, TPR: 0.749, FPR: 0.020, PPV: 0.993, ACC: 0.796,
N N D R_s: P: 398, N: 102, TP: 380, FN: 18, FP: 0, TN: 102, TPR: 0.955, FPR: 0.000, PPV: 1.000, ACC: 0.964,

./bikes/H6p_
FixDist: P: 92, N: 40782, TP: 77, FN: 15, FP: 1888, TN: 38894, TPR: 0.837, FPR: 0.046, PPV: 0.039, ACC: 0.953,
Nearest: P: 92, N: 40782, TP: 77, FN: 15, FP: 1871, TN: 38911, TPR: 0.837, FPR: 0.046, PPV: 0.040, ACC: 0.954,
N N D R: P: 92, N: 40782, TP: 58, FN: 34, FP: 102, TN: 40680, TPR: 0.630, FPR: 0.003, PPV: 0.362, ACC: 0.997,

FixDist_s: P: 389, N: 111, TP: 354, FN: 35, FP: 5, TN: 106, TPR: 0.910, FPR: 0.045, PPV: 0.986, ACC: 0.920,
Nearest_s: P: 389, N: 111, TP: 296, FN: 93, FP: 4, TN: 107, TPR: 0.761, FPR: 0.036, PPV: 0.987, ACC: 0.806,
N N D R_s: P: 389, N: 111, TP: 368, FN: 21, FP: 2, TN: 109, TPR: 0.946, FPR: 0.018, PPV: 0.995, ACC: 0.954,
```

Report 1.1. Feature detection-leuven

Specific: Leuven img *1* –img *6*

Harmonic mean (*All self coded, except basic np. cv2. library*)

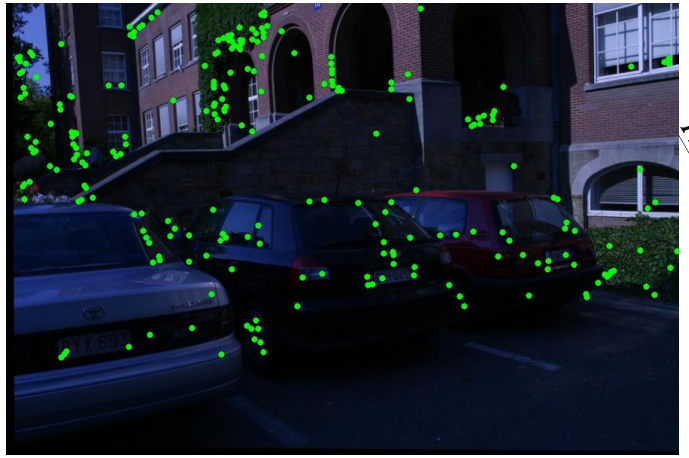
Img2

Img4 H1to4p

Img4 H1to6p

Using SIFT (*involve using a part of cv2.sift library functions*)

Img2



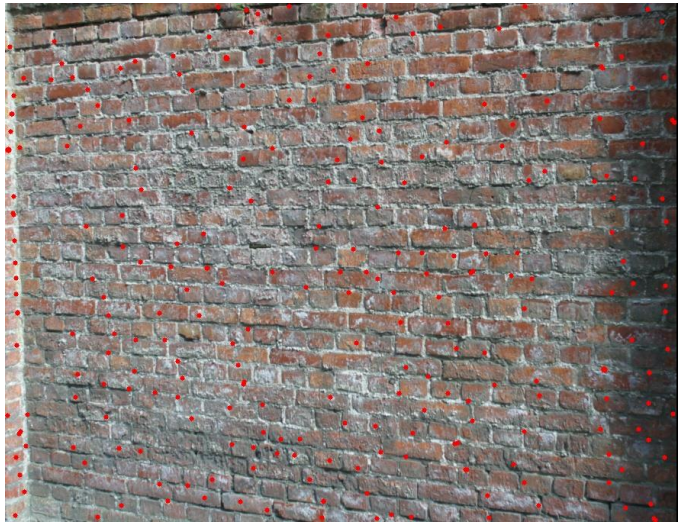
Img4 H1to4p

Img4 H1to6p

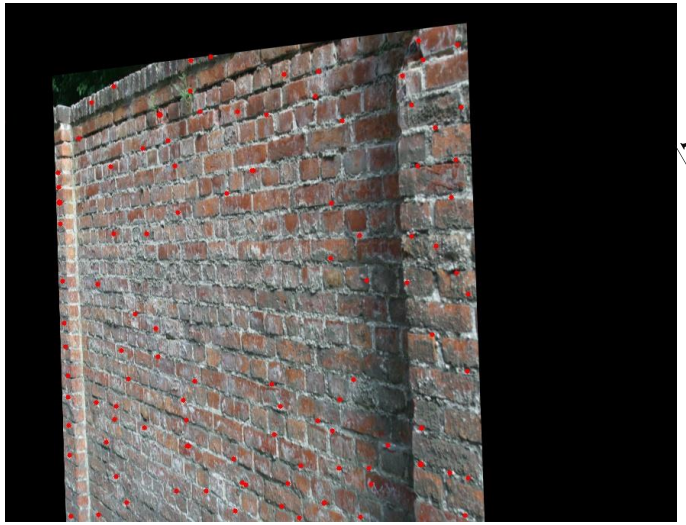
Report 1.2. Feature detection-wall

Specific: wall img *2* –img *4*

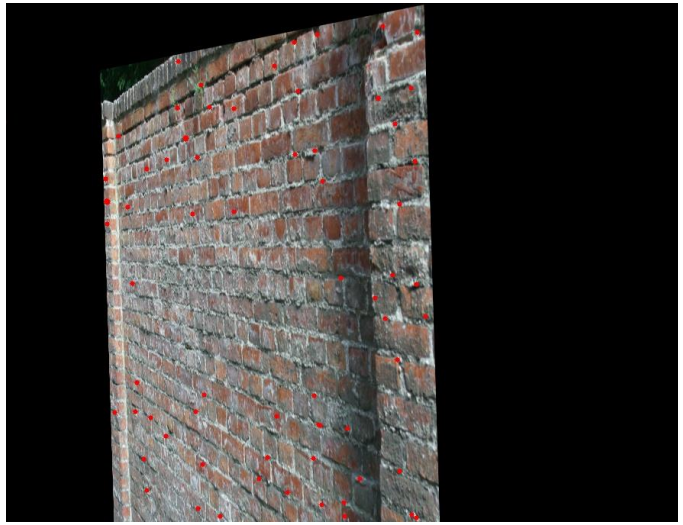
Harmonic mean (*All self coded, except basic np. cv2. library*)



Img2

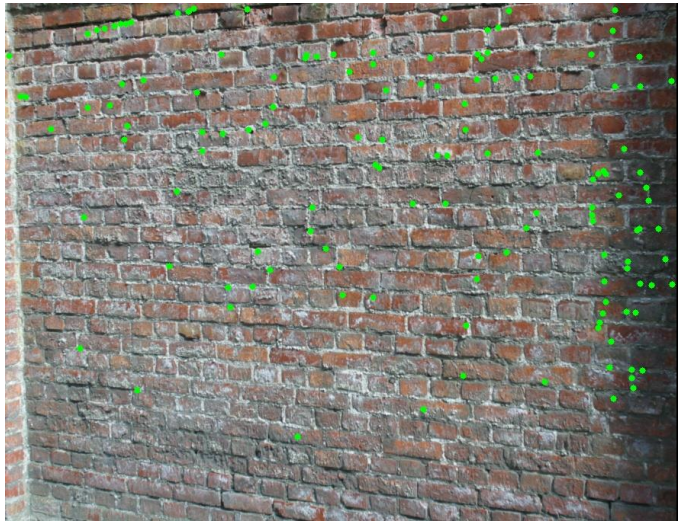


Img4 H1to4p

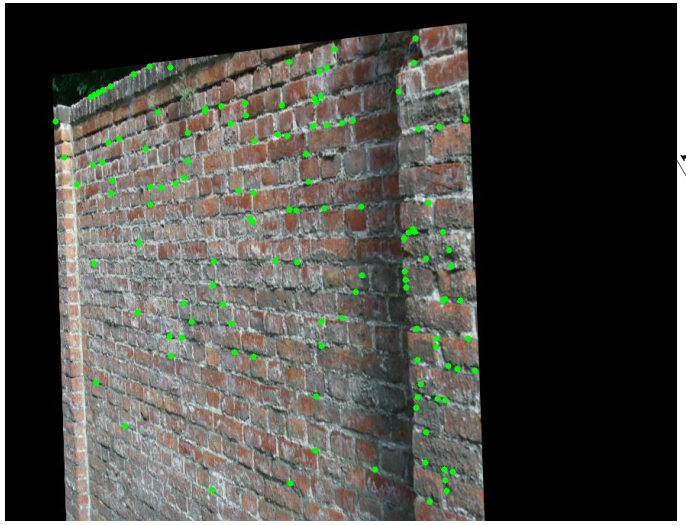


Img4 H1to6p

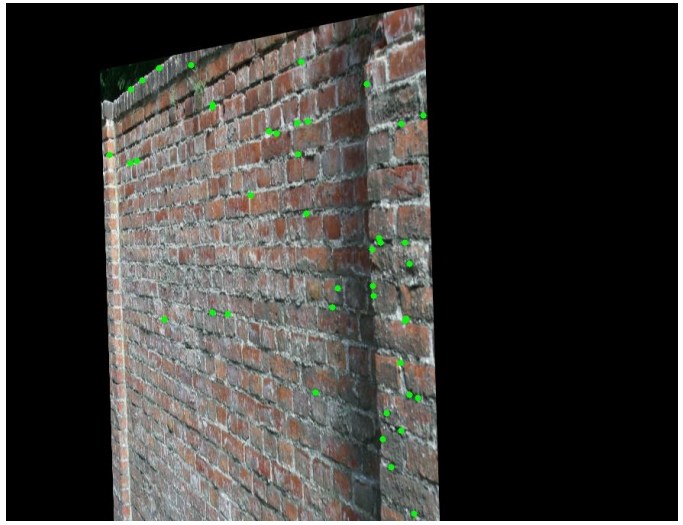
Using SIFT (*involve using a part of cv2.sift library functions*)



Img2



Img4 H1to4p



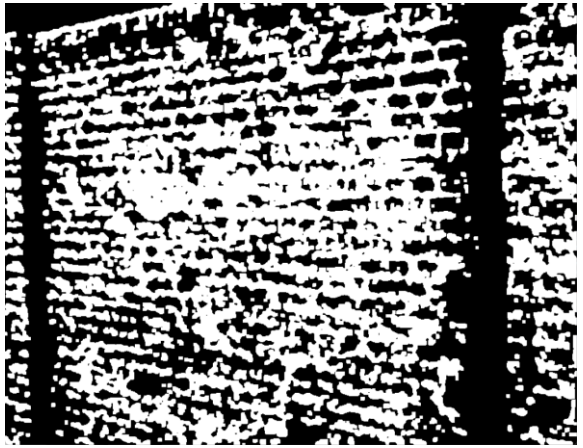
Img4 H1to6p

Report 1. Feature detection-*Highlight*

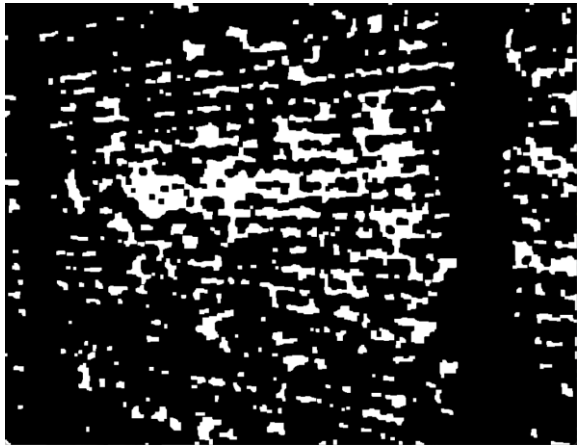
I use Multi Stages Erosion and Dilate, to fit different detail-scale levels of the R matrix

```python
def FeatureMeasure2Points(R, npoints,threshold = 0.01):

    # Find the locations of local maxima above the threshold
    h, w = R.shape
    mask = np.zeros((h, w), np.uint8)
    mask[R > threshold] = 1

    # Use morphological dilation to find the connected regions
    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
    mask_erosion1 = cv2.erode(mask,          kernel,iterations = 2)
    mask_dilate1 = cv2.dilate(mask_erosion1, kernel,iterations = 2)
    mask_erosion2 = cv2.erode(mask_dilate1,  kernel,iterations = 5)
    mask_dilate2 = cv2.dilate(mask_erosion2, kernel,iterations = 2)
    mask_erosion3 = cv2.erode(mask_dilate2,  kernel,iterations = 5)
    mask_dilate3 = cv2.dilate(mask_erosion3, kernel,iterations = 2)
    mask_erosion4 = cv2.erode(mask_dilate3,  kernel,iterations = 5)
    mask_dilate4 = cv2.dilate(mask_erosion4, kernel,iterations = 2)

    #cv2.imshow('imgh',mask_dilate1*250)
    # Compute the centroid of each connected region as the feature point location
    num_labels, labels, stats1, centroids1 = cv2.connectedComponentsWithStats(mask_dilate1)
    num_labels, labels, stats2, centroids2 = cv2.connectedComponentsWithStats(mask_dilate2)
    num_labels, labels, stats3, centroids3 = cv2.connectedComponentsWithStats(mask_dilate3)
    num_labels, labels, stats4, centroids4 = cv2.connectedComponentsWithStats(mask_dilate4)

    # Remove the background label

    centroids_=np.concatenate((centroids1[1:],centroids2[1:],centroids3[1:],centroids4[1:]), axis=0)
    stats_=np.concatenate((stats1[1:],stats2[1:],stats3[1:],stats4[1:]), axis=0)
    stats_=stats_[:,4]
    # Sample a fixed number of feature points from the list of all feature points
    centroids=[]
    for idx in range(len(stats_)):
        if 100 < stats_[idx] < 500:
            centroids=np.concatenate((centroids, centroids_[idx]), axis=0)
    centroids=np.reshape(centroids, (int(len(centroids)/2),2))

    if len(centroids) > npoints:
        centroids = centroids[:npoints]

    # Return the x, y coordinates of the feature points and the mask of the detected regions
    x = centroids[:, 0]
    y = centroids[:, 1]
    mask = np.zeros_like(R)
    for point in centroids:
        mask[int(point[1]), int(point[0])] = 1

    return x, y, mask
```

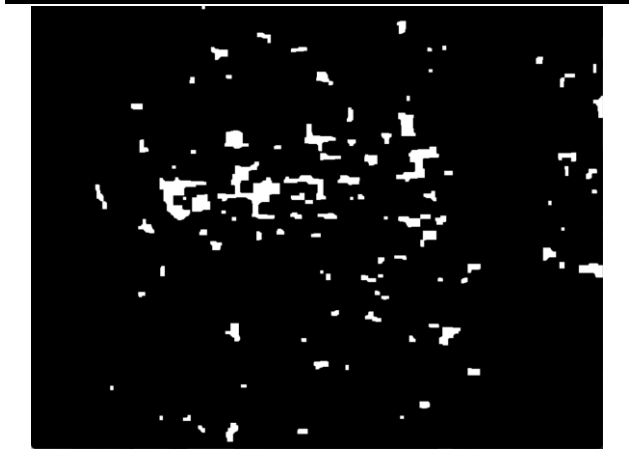
```python
mask_erosion1 = cv2.erode(mask,          kernel,iterations = 2)
mask_dilate1 = cv2.dilate(mask_erosion1, kernel,iterations = 2)
```



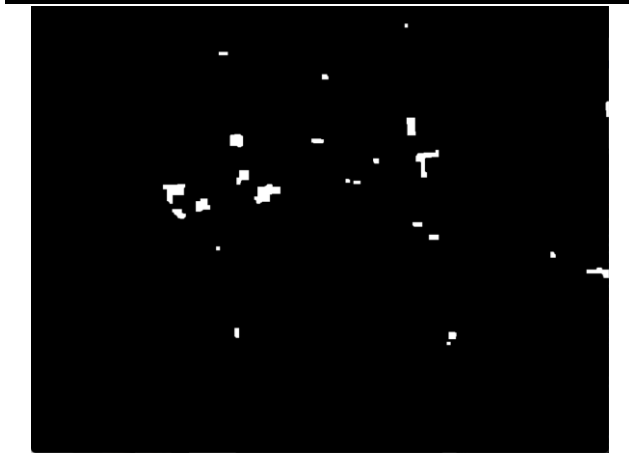
```python
mask_erosion2 = cv2.erode(mask_dilate1,  kernel,iterations = 5)
mask_dilate2 = cv2.dilate(mask_erosion2, kernel,iterations = 2)
```



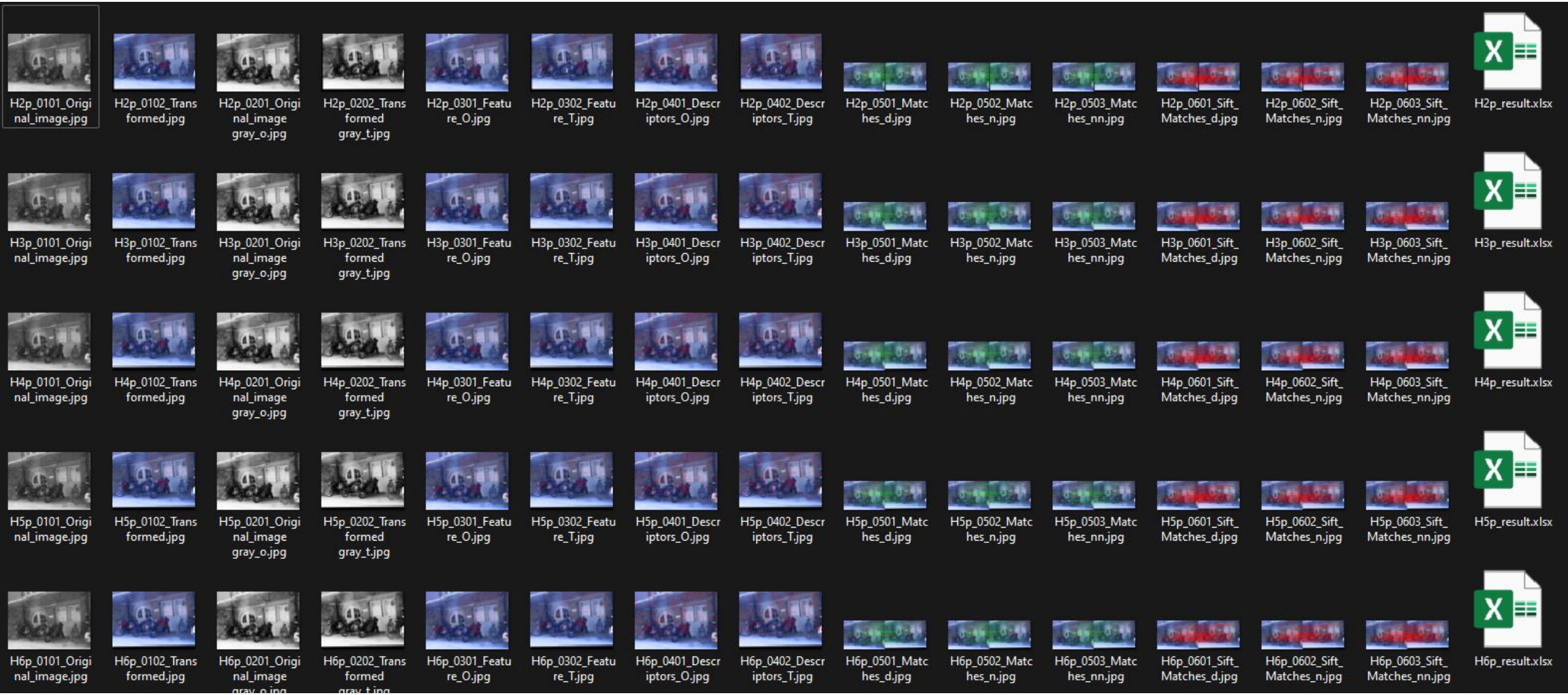
```python
mask_erosion3 = cv2.erode(mask_dilate2,  kernel,iterations = 5)
mask_dilate3 = cv2.dilate(mask_erosion3, kernel,iterations = 2)
```



```python
mask_erosion4 = cv2.erode(mask_dilate3,  kernel,iterations = 5)
mask_dilate4 = cv2.dilate(mask_erosion4, kernel,iterations = 2)
```



Report 2.1. Feature matching and evaluation-Bikes



Report 2.1. Feature matching and evaluation-Bikes

Specific: H1to*4*p over img *1* –img *3*

Harmonic mean (*All self coded, except basic np. cv2. library*)

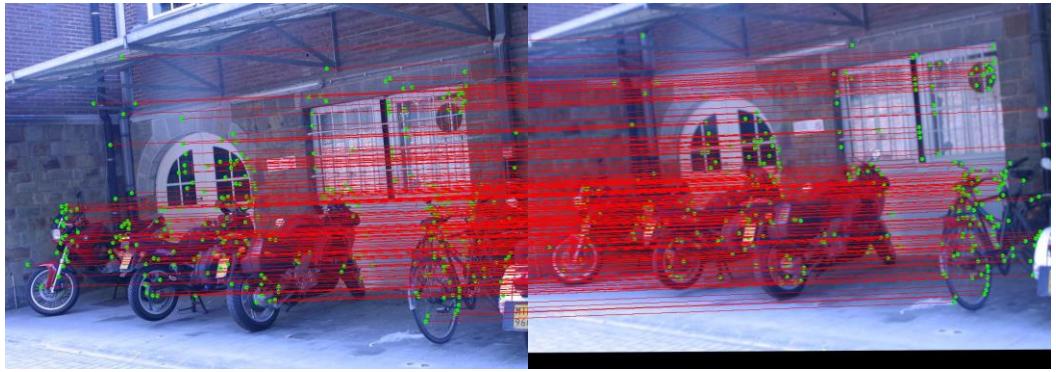
DistThresh

NearestMatch

NearestRatio

* *Red Dots* are Keypoints, *Green Lines* are Ture Positive Matches, bad results are not shown (they are *Red Dots* without *matching*).

Using SIFT (*involve using a part of cv2.sift library functions*)



DistThresh

NearestMatch

NearestRatio

* *Green Dots* are Keypoints, *Red Lines* are Ture Positive Matches, bad results are not shown (they are *Red Dots* without *matching*).

Report 2.1. Feature matching and evaluation-Bikes

Specific: H1to*6*p over img *1* –img *3*

Harmonic mean (*All self coded, except basic np. cv2. library*)

DistThresh

NearestMatch

NearestRatio

* *Red Dots* are Keypoints, *Green Lines* are Ture Positive Matches, bad results are not shown (they are *Red Dots* without *matching*).

Using SIFT (*involve using a part of cv2.sift library functions*)

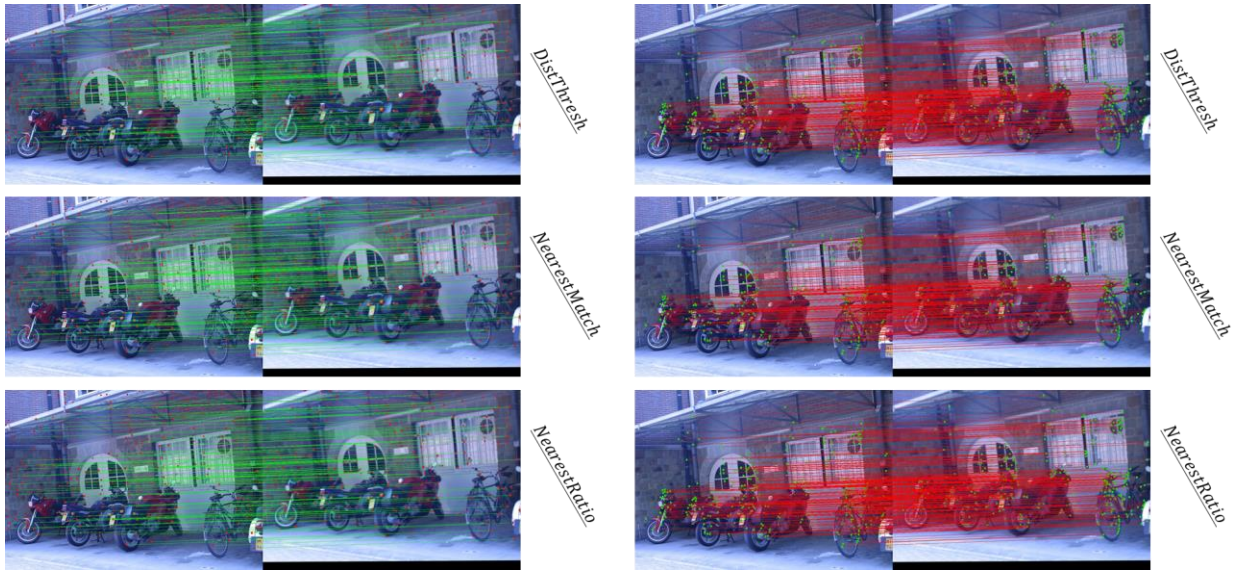
DistThresh

NearestMatch

NearestRatio

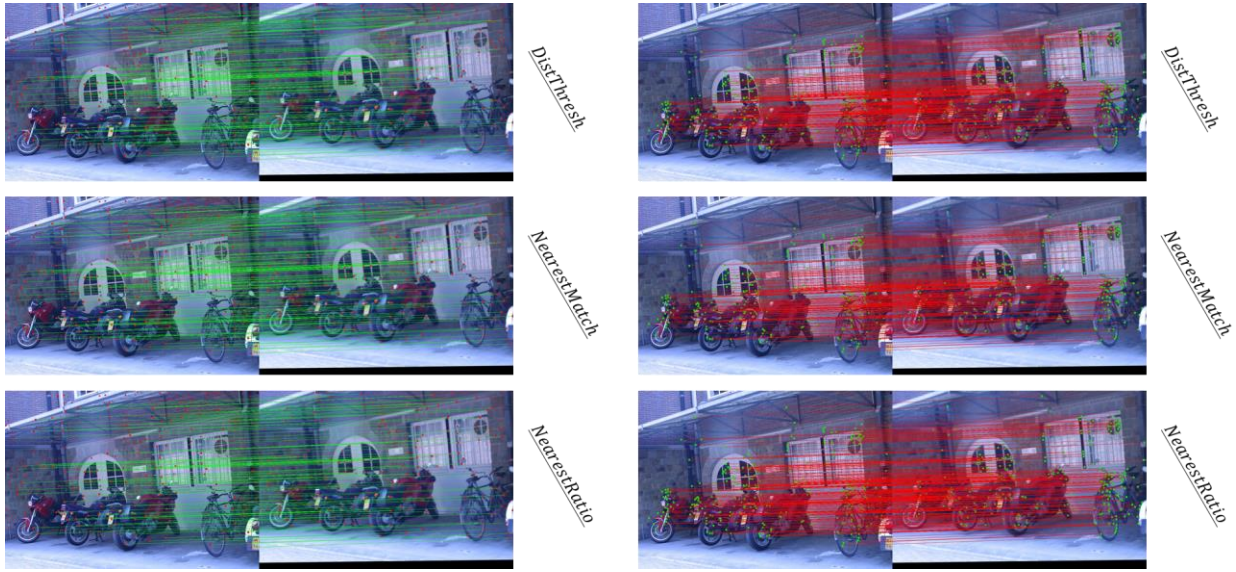
* *Green Dots* are Keypoints, *Red Lines* are Ture Positive Matches, bad results are not shown (they are *Red Dots* without *matching*).

Report 2.1. Feature matching and evaluation-Bikes



Specific: H1to*4*p over img *1* –img *3*

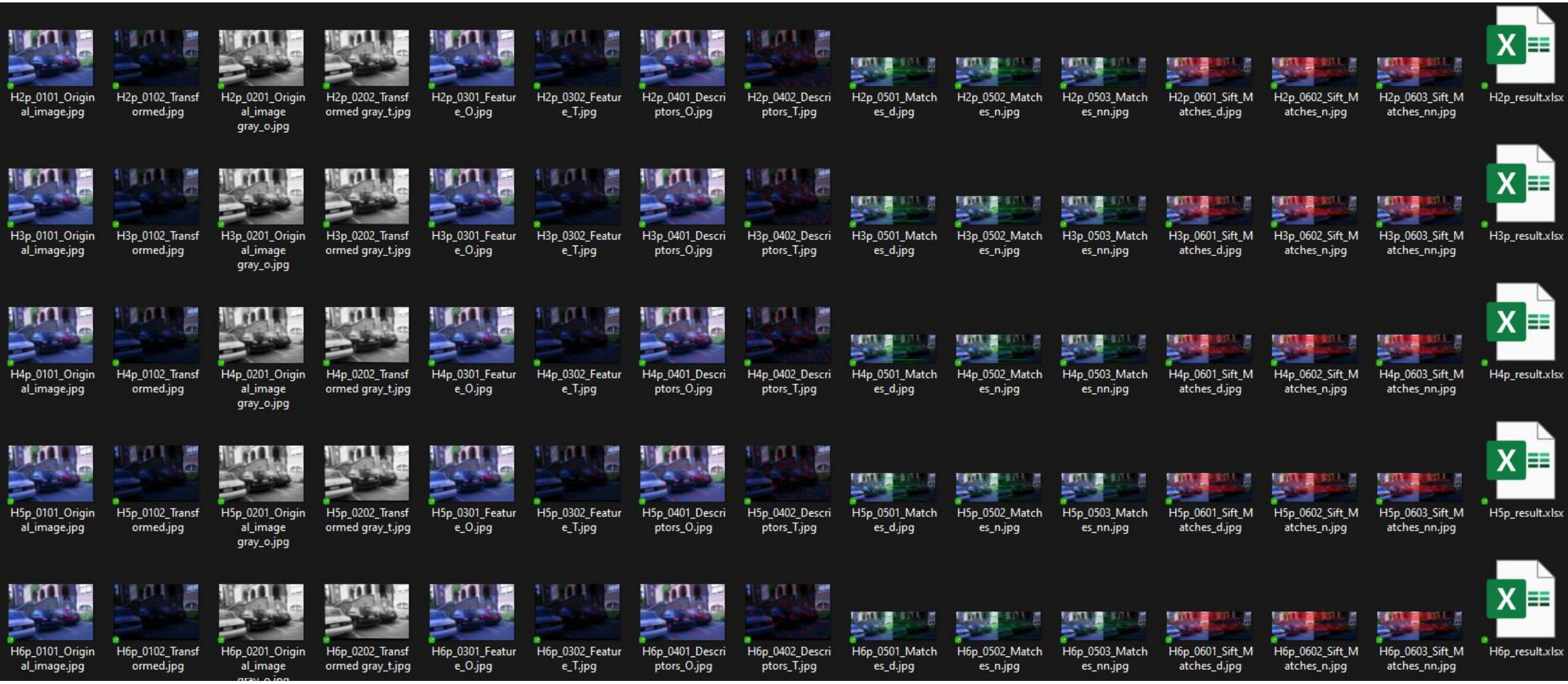
	FixDist	Nearest	N N D R	FixDistS	NearestS	N N D RS
P	95	95	95	389	389	389
N	42919	42919	42919	111	111	111
TP	78	78	64	358	296	379
FN	17	17	31	31	93	10
FP	2133	1972	99	7	4	1
TN	40786	40947	42820	104	107	110
TPR	82.11%	82.11%	67.37%	92.03%	76.09%	97.43%
FPR	4.97%	4.59%	0.23%	6.31%	3.60%	0.90%
PPV	3.53%	3.80%	39.26%	98.08%	98.67%	99.74%
ACC	95.00%	95.38%	99.70%	92.40%	80.60%	97.80%



Specific: H1to*6*p over img *1* –img *3*

	FixDist	Nearest	N N D R	FixDistS	NearestS	N N D RS
P	92	92	92	389	389	389
N	40782	40782	40782	111	111	111
TP	77	77	58	354	296	368
FN	15	15	34	35	93	21
FP	1888	1871	102	5	4	2
TN	38894	38911	40680	106	107	109
TPR	83.70%	83.70%	63.04%	91.00%	76.09%	94.60%
FPR	4.63%	4.59%	0.25%	4.50%	3.60%	1.80%
PPV	3.92%	3.95%	36.25%	98.61%	98.67%	99.46%
ACC	95.34%	95.39%	99.67%	92.00%	80.60%	95.40%

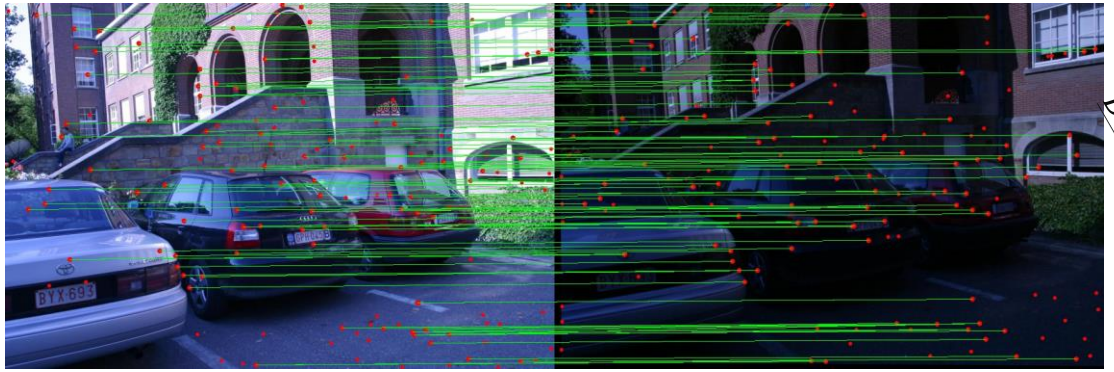
Report 2.2. Feature matching and evaluation-Leuven



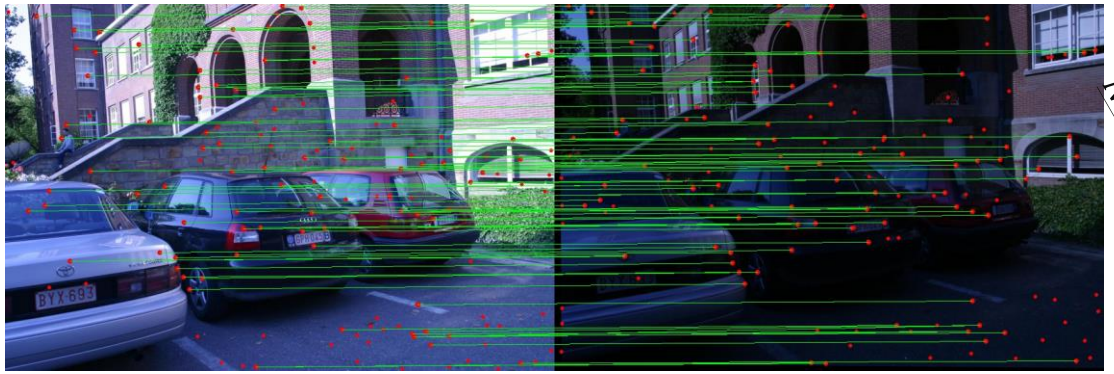
Report 2.2. Feature matching and evaluation-Leuven

Specific: H1to*4*p over img *1* –img *6*

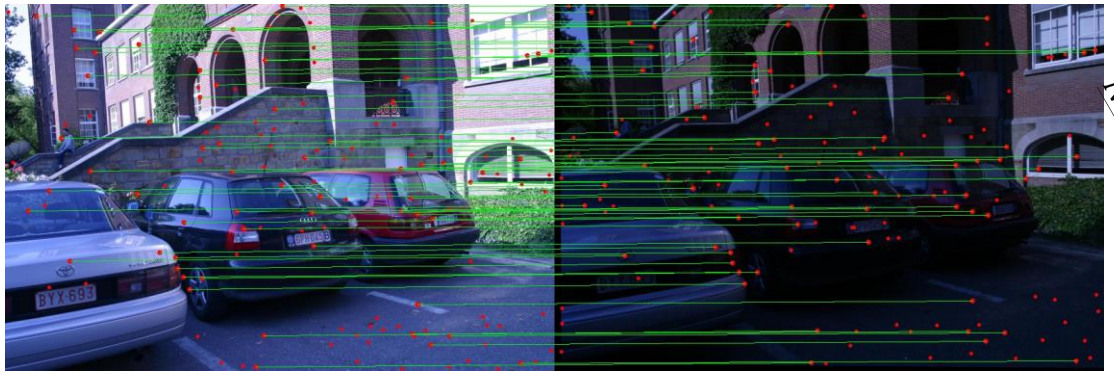
Harmonic mean (*All self coded, except basic np. cv2. library*)



DistThresh



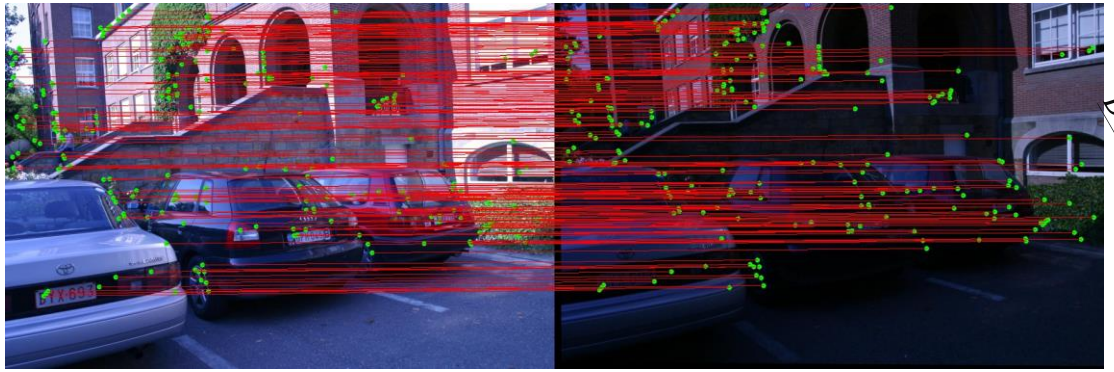
NearestMatch



NearestRatio

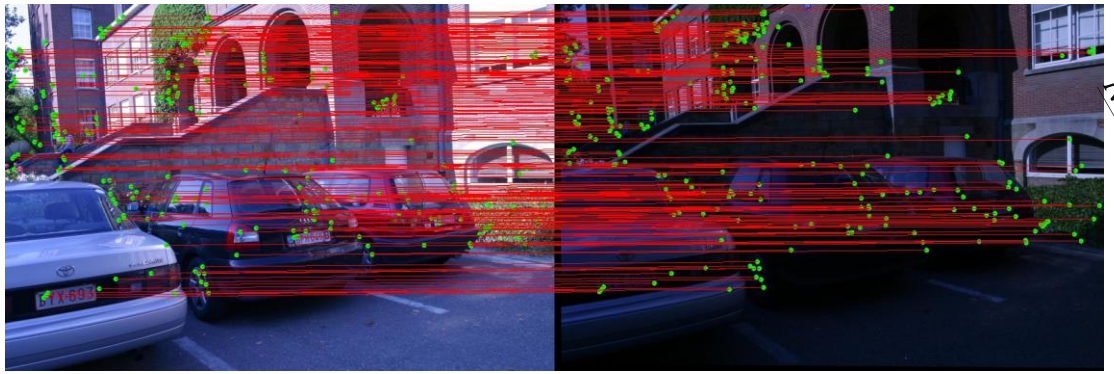
* *Red Dots* are Keypoints, *Green Lines* are Ture Positive Matches, bad results are not shown (they are *Red Dots* without *matching*).

Using SIFT (*involve using a part of cv2.sift library functions*)



DistThresh

NearestMatch



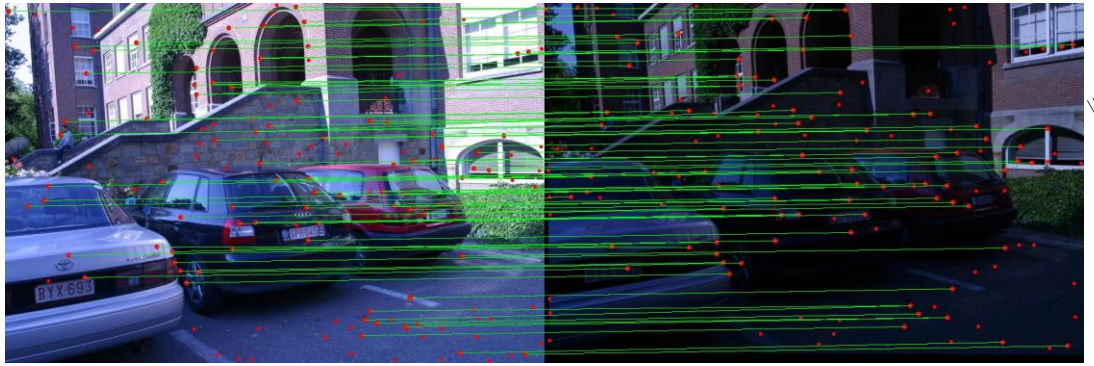
NearestRatio

* *Green Dots* are Keypoints, *Red Lines* are Ture Positive Matches, bad results are not shown (they are *Red Dots* without *matching*).

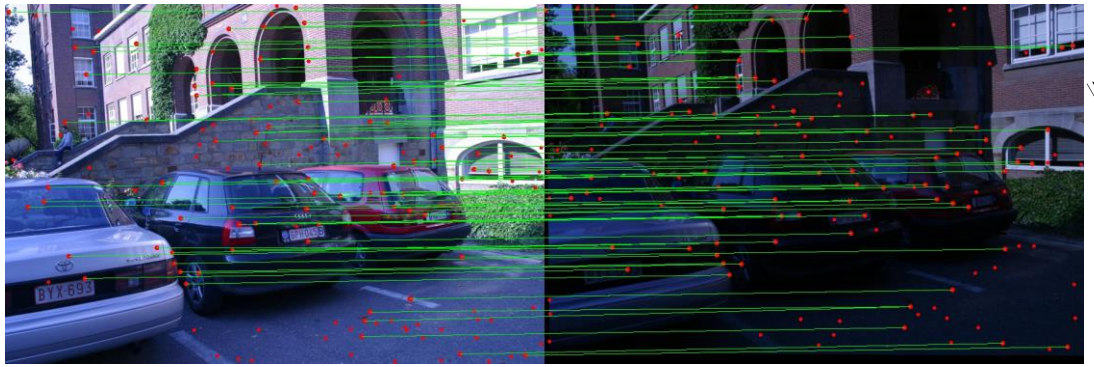
Report 2.2. Feature matching and evaluation-Leuven

Specific: H1to*6*p over img *1* –img *6*

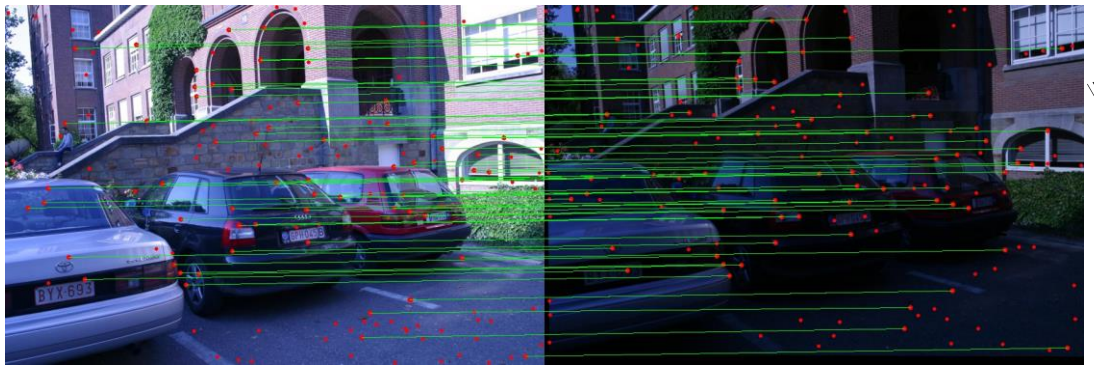
Harmonic mean (*All self coded, except basic np. cv2. library*)



DistThresh



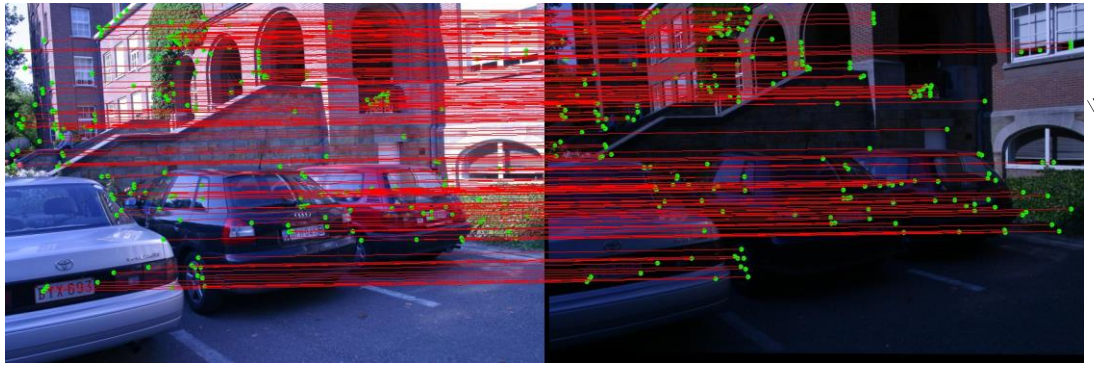
NearestMatch



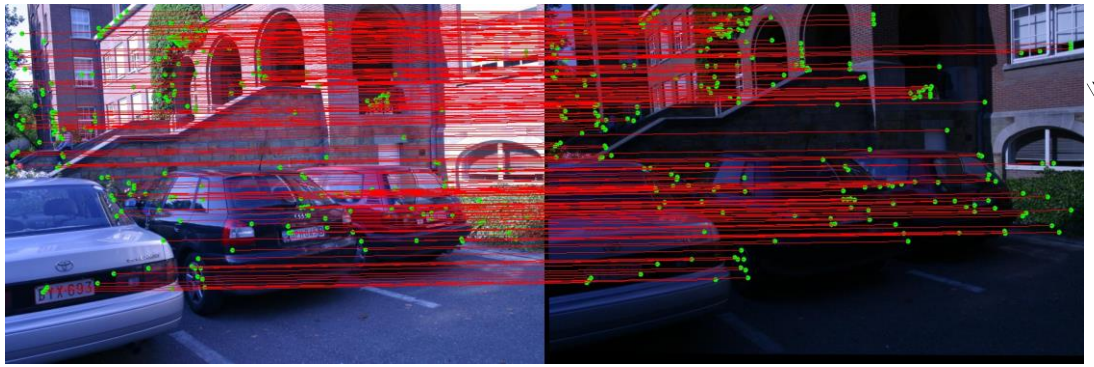
NearestRatio

* *Red Dots* are Keypoints, *Green Lines* are Ture Positive Matches, bad results are not shown (they are *Red Dots* without *matching*).

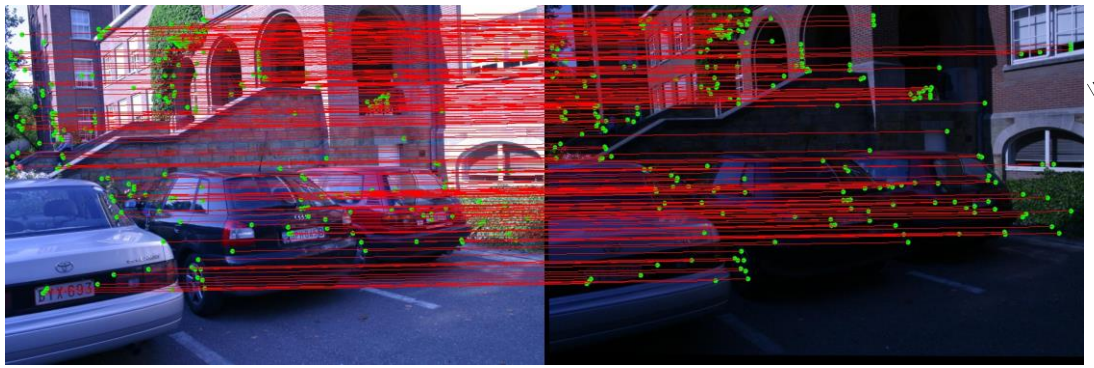
Using SIFT (*involve using a part of cv2.sift library functions*)



DistThresh



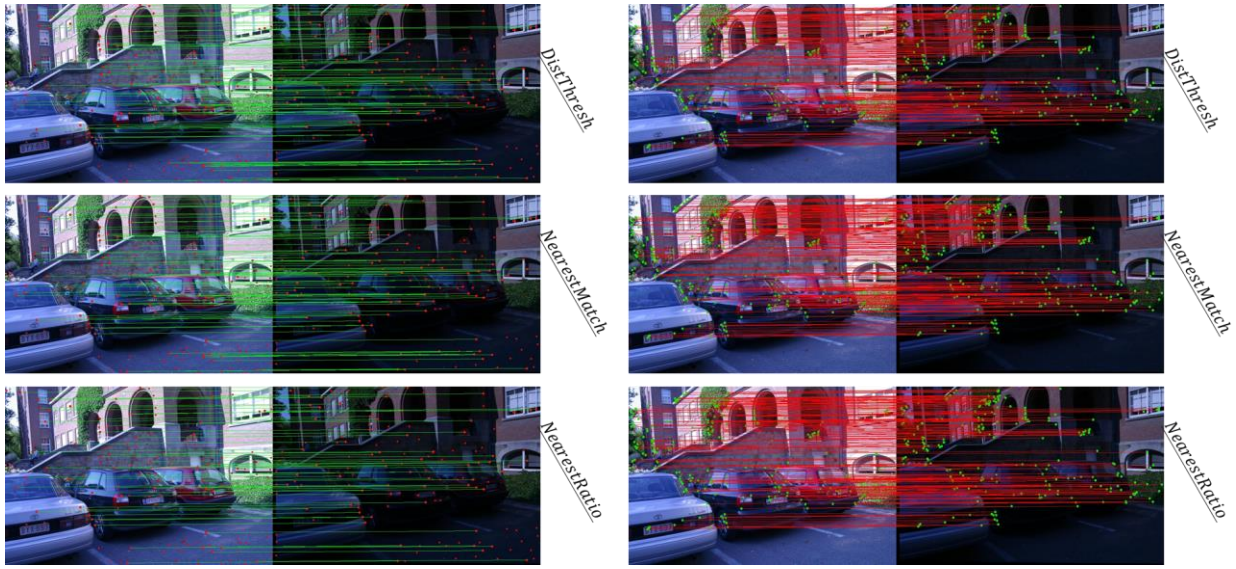
NearestMatch



NearestRatio

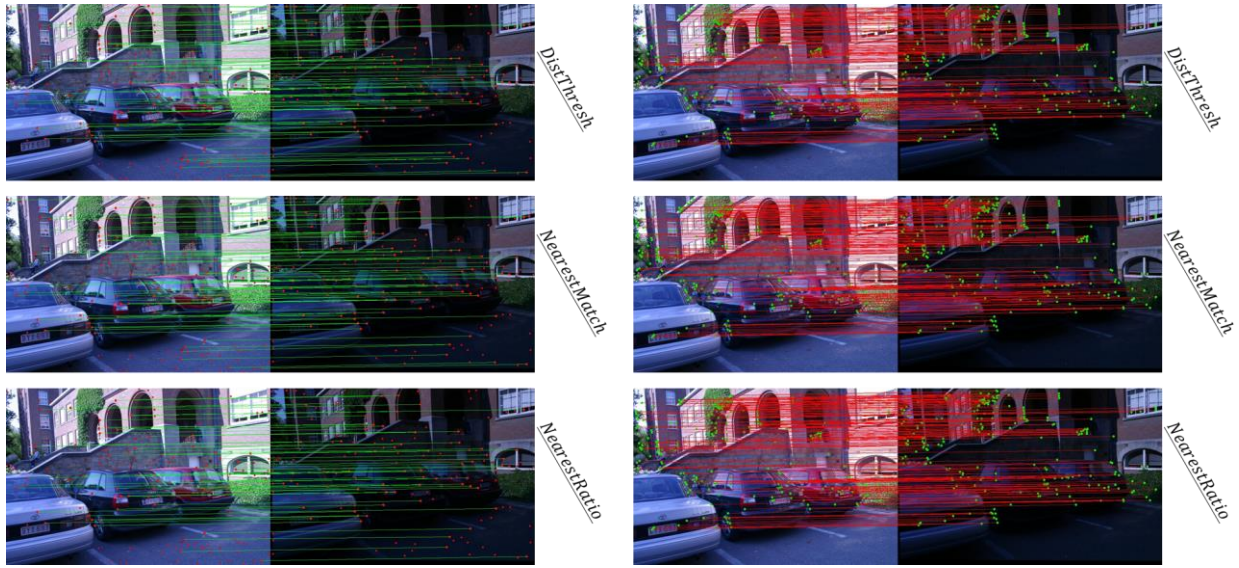
* *Green Dots* are Keypoints, *Red Lines* are Ture Positive Matches, bad results are not shown (they are *Red Dots* without *matching*).

Report 2.2. Feature matching and evaluation-Leuven



Specific: H1to4p over img 1 –img 6

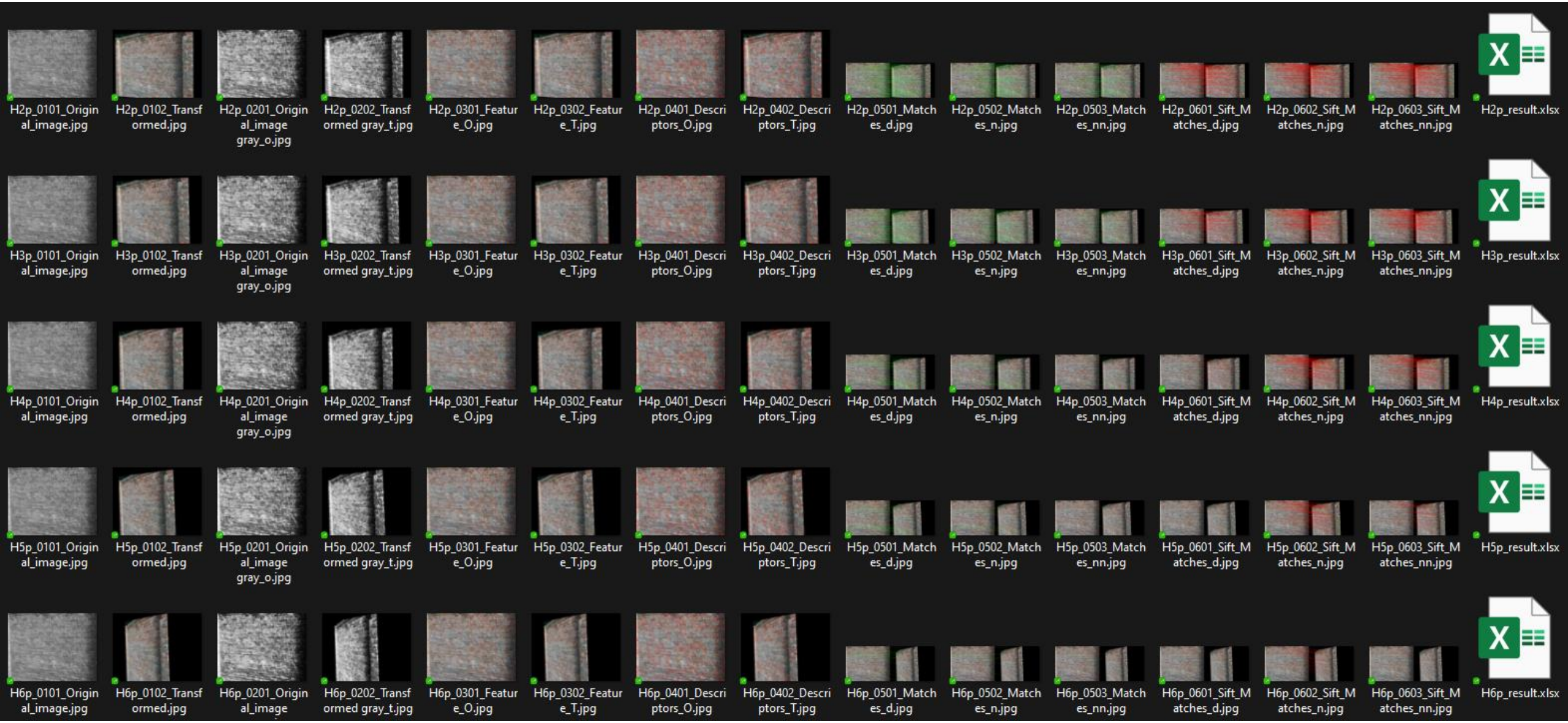
	FixDist	Nearest	N N D R	FixDist Sift	Nearest Sift	N N D R Sift
P	82	82	82	316	316	316
N	20654	20654	20654	184	184	184
TP	51	48	33	243	287	287
FN	31	34	49	73	29	29
FP	1731	1420	70	1	13	2
TN	18923	19234	20584	183	171	182
TPR	62.20%	58.54%	40.24%	76.90%	90.82%	90.82%
FPR	8.38%	6.88%	0.34%	0.54%	7.07%	1.09%
PPV	2.86%	3.27%	32.04%	99.59%	95.67%	99.31%
ACC	91.50%	92.99%	99.43%	85.20%	91.60%	93.80%



Specific: H1to6p over img 1 –img 6

	FixDist	Nearest	N N D R	FixDist Sift	Nearest Sift	N N D R Sift
P	77	77	77	304	304	304
N	20515	20515	20515	196	196	196
TP	45	43	31	244	283	274
FN	32	34	46	60	21	30
FP	2050	1415	62	0	17	1
TN	18465	19100	20453	196	179	195
TPR	58.44%	55.84%	40.26%	80.26%	93.09%	90.13%
FPR	9.99%	6.90%	0.30%	0.00%	8.67%	0.51%
PPV	2.15%	2.95%	33.33%	100.00%	94.33%	99.64%
ACC	89.89%	92.96%	99.48%	88.00%	92.40%	93.80%

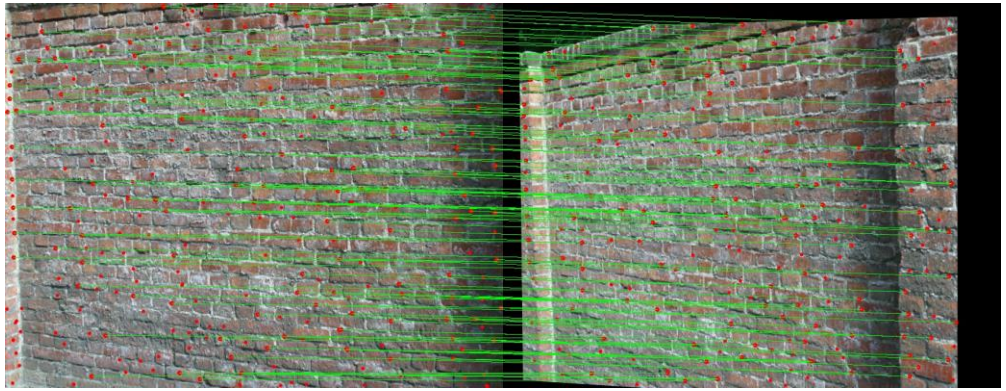
Report 2.3. Feature matching and evaluation-Wall



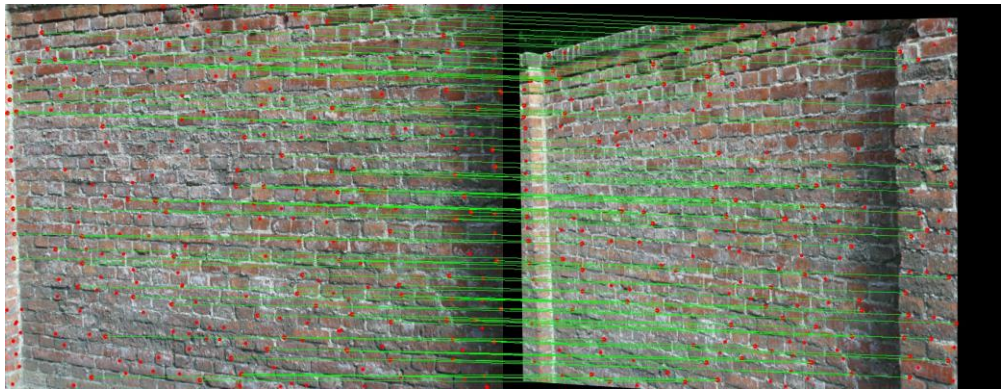
Report 2.3. Feature matching and evaluation-Wall

Specific: H1to2p over img 2 –img 4

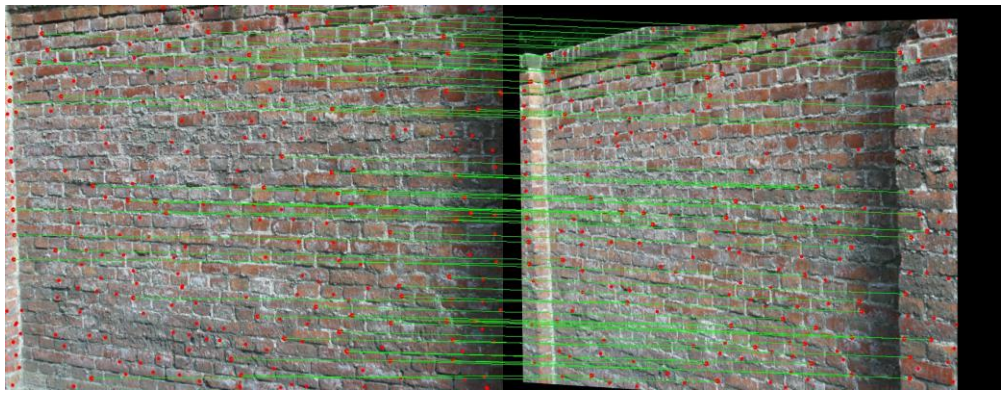
Harmonic mean (*All self coded, except basic np. cv2. library*)



DistThresh



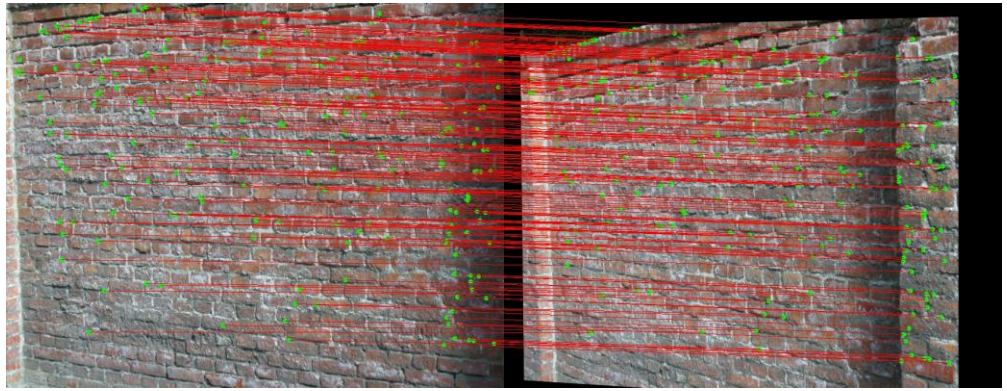
NearestMatch



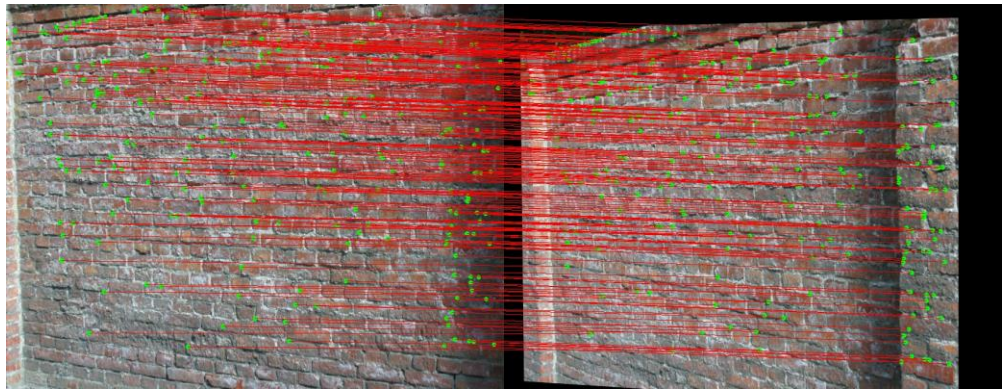
NearestRatio

* *Red Dots* are Keypoints, *Green Lines* are Ture Positive Matches, bad results are not shown (they are *Red Dots* without *matching*).

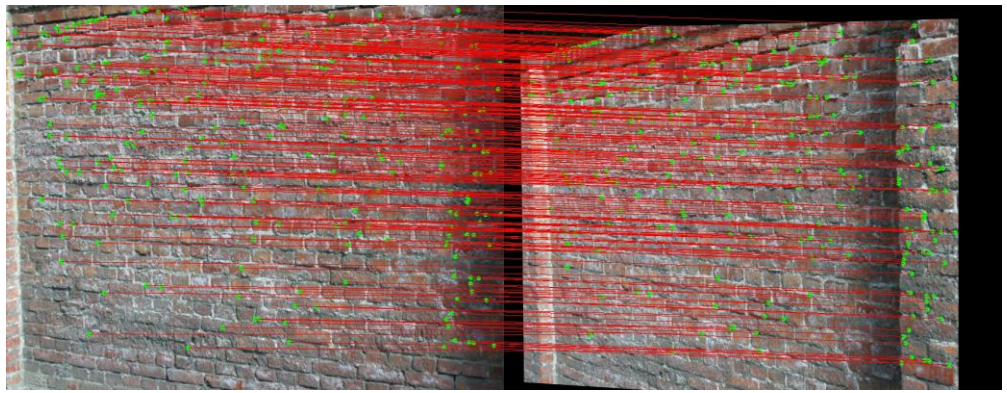
Using SIFT (*involve using a part of cv2.sift library functions*)



DistThresh



NearestMatch



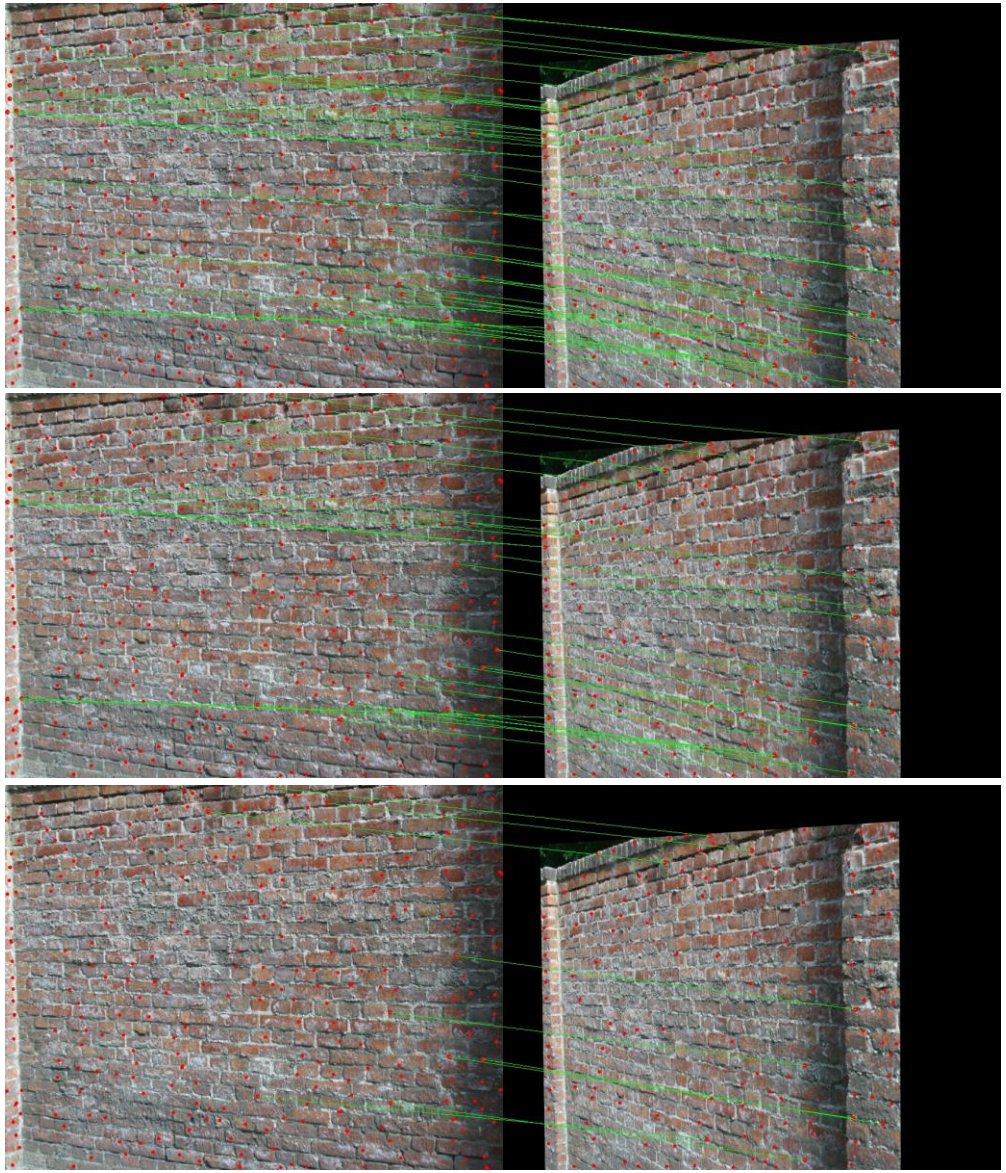
NearestRatio

* *Green Dots* are Keypoints, *Red Lines* are Ture Positive Matches, bad results are not shown (they are *Red Dots* without *matching*).

Report 2.3. Feature matching and evaluation-Wall

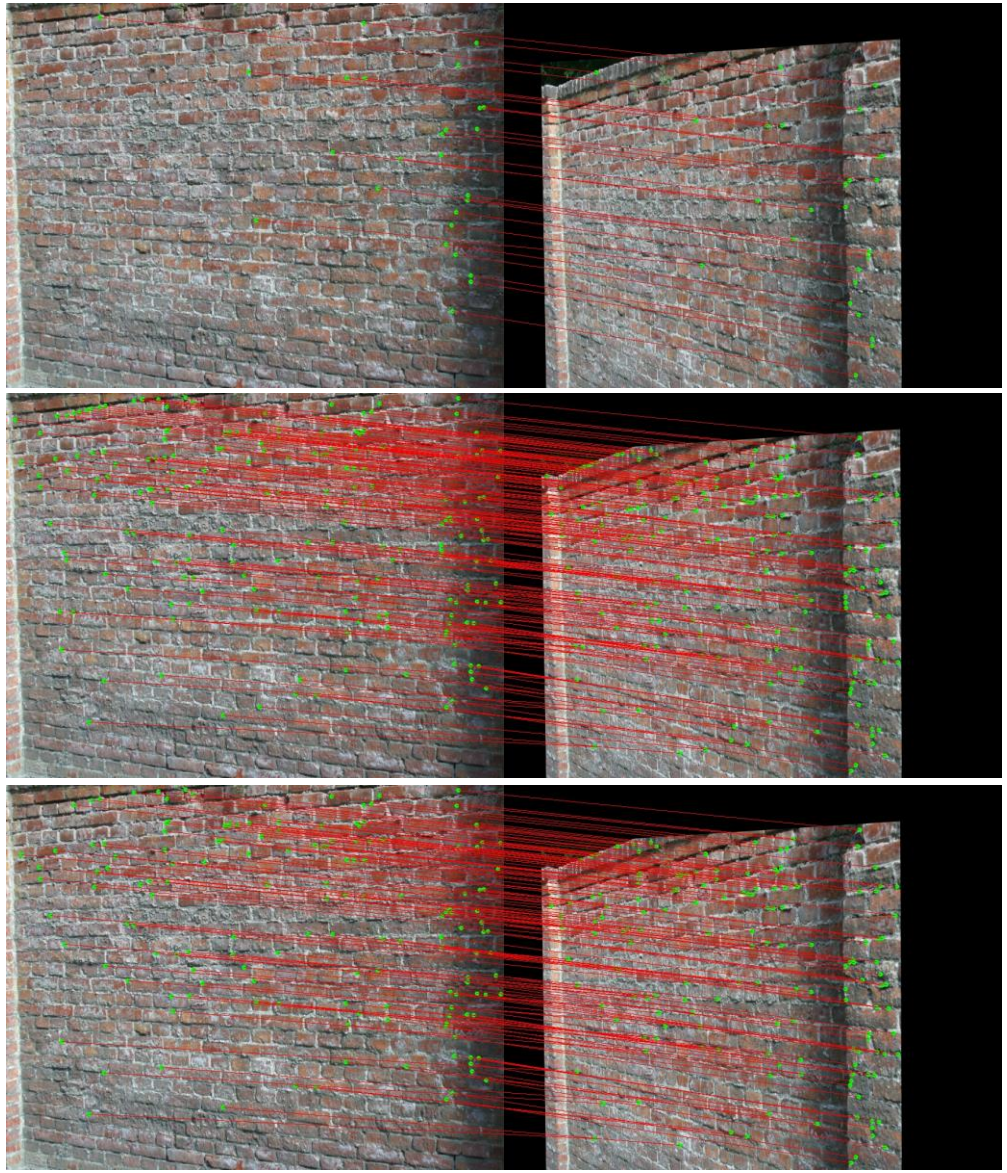
Specific: H1to*4*p over img *2* –img *4*

Harmonic mean (*All self coded, except basic np. cv2. library*)



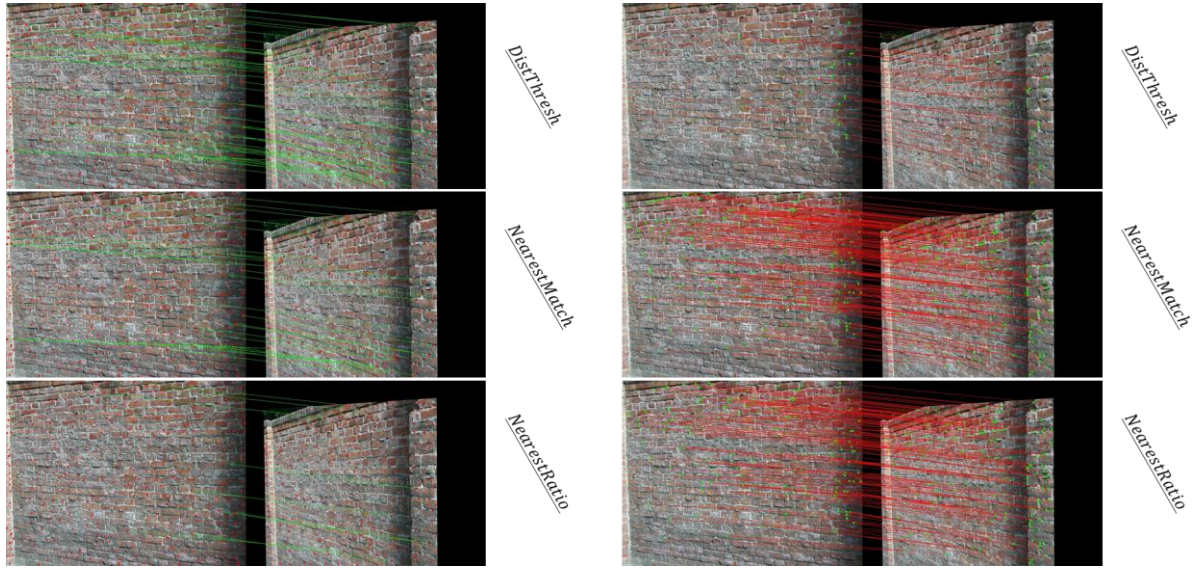
* *Red Dots* are Keypoints, *Green Lines* are Ture Positive Matches, bad results are not shown (they are *Red Dots* without *matching*).

Using SIFT (*involve using a part of cv2.sift library functions*)



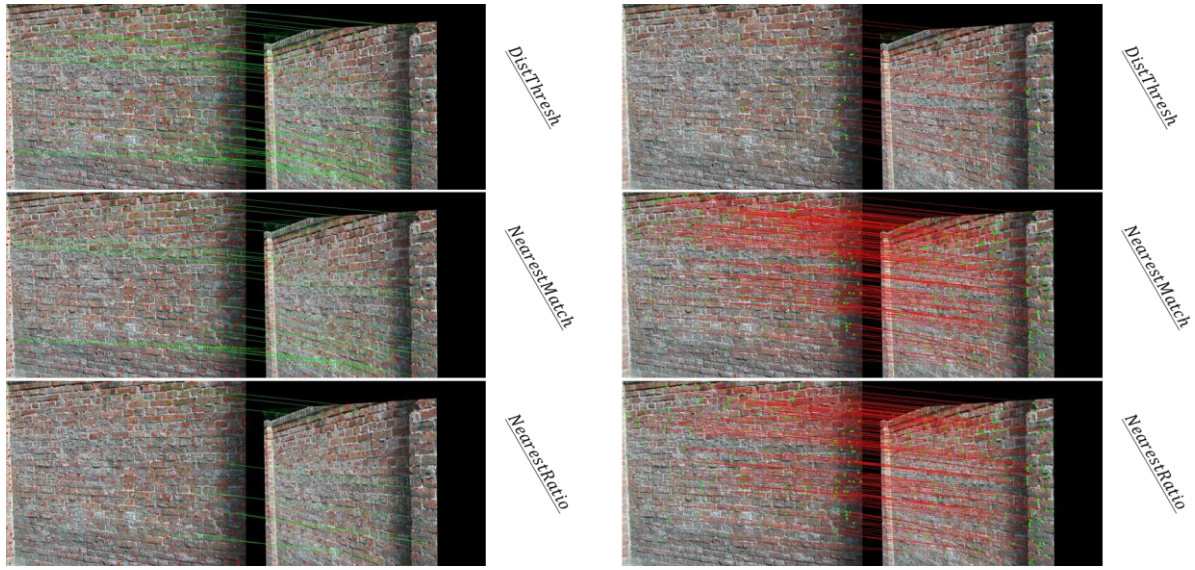
* *Green Dots* are Keypoints, *Red Lines* are Ture Positive Matches, bad results are not shown (they are *Red Dots* without *matching*).

Report 2.3. Feature matching and evaluation-Wall



Specific: H1to2p over img 2 –img 4

	FixDist	Nearest	N N D R	FixDistS	NearestS	N N D RS
P	85	85	85	334	334	334
N	53420	53420	53420	166	166	166
TP	62	50	32	253	300	318
FN	23	35	53	81	34	16
FP	5949	2041	96	0	0	0
TN	47471	51379	53324	166	166	166
TPR	72.94%	58.82%	37.65%	75.75%	89.82%	95.21%
FPR	11.14%	3.82%	0.18%	0.00%	0.00%	0.00%
PPV	1.03%	2.39%	25.00%	100.00%	100.00%	100.00%
ACC	88.84%	96.12%	99.72%	83.80%	93.20%	96.80%



Specific: H1to4p over img 2 –img 4

	FixDist	Nearest	N N D R	FixDistS	NearestS	N N D RS
P	56	56	56	235	235	235
N	34918	34918	34918	265	265	265
TP	23	13	4	29	231	181
FN	33	43	52	206	4	54
FP	5728	1353	114	0	69	0
TN	29190	33565	34804	265	196	265
TPR	41.07%	23.21%	7.14%	12.34%	98.30%	77.02%
FPR	16.40%	3.87%	0.33%	0.00%	26.04%	0.00%
PPV	0.40%	0.95%	3.39%	100.00%	77.00%	100.00%
ACC	83.53%	96.01%	99.53%	58.80%	85.40%	89.20%

Interpretation & Discussion

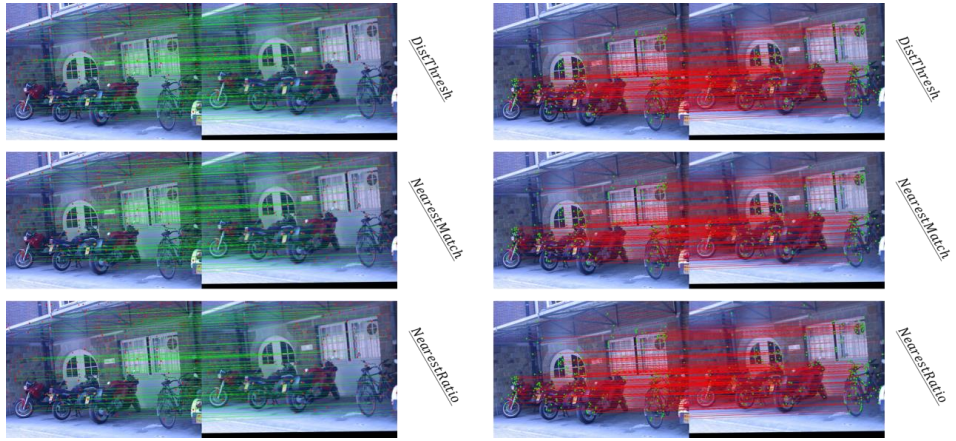
This assignment is making hands-on test with feature detection, description, and matching, which are fundamental concepts in computer vision. By implementing the pipeline, we can learn the practical aspects of these concepts and gain insight into how the CV functions work in different scenarios.

The first step of the pipeline involves feature detection using autocorrelation matrices and scalar feature detection measures. This is an important step as it determines which points in the image are likely to be good features for matching. By visualizing the detected feature points, students can gain a better understanding of how the algorithm works and what kind of features are being detected.

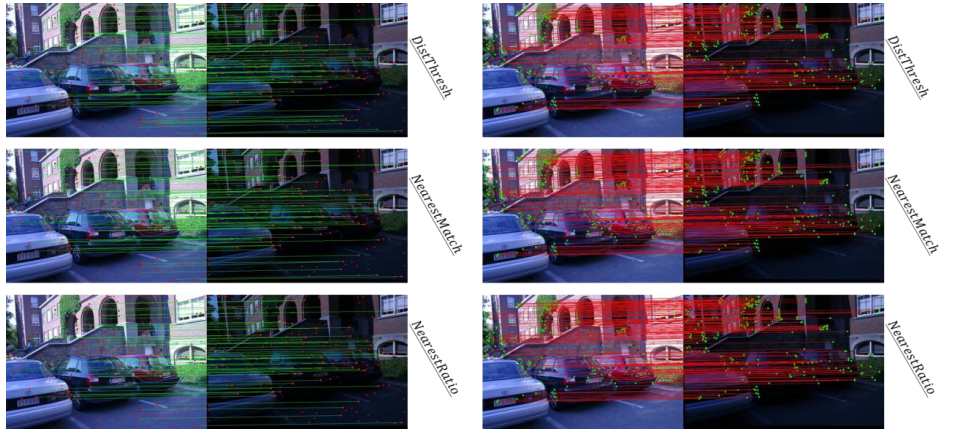
The second step of the pipeline involves feature matching and evaluation using a feature descriptor. This is an essential step in many computer vision applications, such as object recognition and image registration. By comparing the performance of different matching strategies and evaluating the results using homography matrices, students can gain an understanding of the strengths and limitations of different feature descriptors and matching algorithms.

Overall, this assignment provides a full pipeline to feature detection, description, and matching in CV. I tried different matching strategies and evaluate their performance, some of them are useful for giving more advanced results.

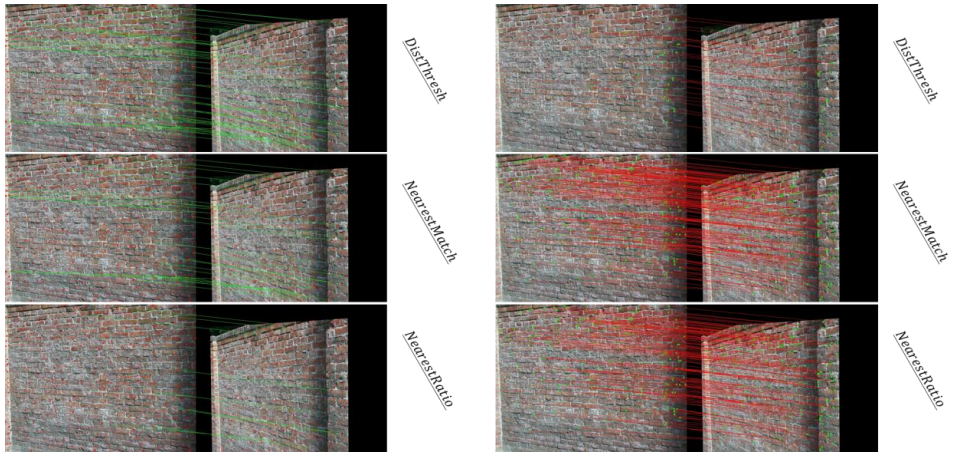
Bikes
Img 1 – 3
①



Leuven
Img 1 – 6
②



Wall
Img 2 – 4
③



Untitled-1

February 24, 2023

[ ]:
```
import numpy as np
from datetime import datetime
import time
import matplotlib.pyplot as plt
import cv2
from PIL import Image
from scipy.signal import argrelmax
import pandas as pd
```

[ ]:
```
marker_color = [0, 0, 255]

def get_current_time():
    current_time = datetime.now().strftime("%H:%M:%S")
    return current_time

folder = "./wall/"
#folder = "./bikes/"
image_id1 = "img2.ppm"
image_id2 = "img4.ppm"
image_id3 = "img4.ppm"

print(get_current_time())
```

[ ]:
```
def read_ppm_file(file_path):
    with open(file_path, 'rb') as f:
        # Read the header information of the PPM file
        header = f.readline().decode('utf-8').strip()
        width, height = map(int, f.readline().decode('utf-8').split())
        max_val = int(f.readline().decode('utf-8'))

        # Read the image data from the PPM file
        img_data = f.read()

        # Convert the image data to a NumPy array
        img_array = bytearray(img_data)
        img_np = np.array(img_array)
```

```
        # Reshape the NumPy array into a 3D array with dimensions (height,␣
 ↪width, 3)
        img_np = img_np.reshape((height, width, 3))

        # Convert the NumPy array to a cv2 object in BGR format
        img_cv2 = cv2.cvtColor(img_np, cv2.COLOR_RGB2BGR)

        return img_cv2

def unify_contrast_brightness(image):
    # Compute the minimum and maximum pixel values in the image
    min_val, max_val, _, _ = cv2.minMaxLoc(image)

    # Normalize the pixel values to span the full range of 0 to 255
    if max_val > min_val:
        normalized = ((image - min_val) / (max_val - min_val)) * 255
    else:
        normalized = image.copy()

    # Convert the pixel values to the range of 0 to 255
    normalized = np.uint8(normalized)

    # Apply a histogram equalization to improve contrast
    equalized = cv2.equalizeHist(normalized)   # type: ignore

    return equalized

def unify_image(img, clip_limit=2.0, tile_size=8, brightness=128, contrast=1.0,␣
 ↪detail_level=1.0):
    # Convert to LAB color space
    lab = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)

    # Split channels
    l, a, b = cv2.split(lab)

    # Apply CLAHE to the L channel
    clahe = cv2.createCLAHE(clipLimit=clip_limit, tileGridSize=(tile_size,␣
 ↪tile_size))
    l = clahe.apply(l)

    # Apply brightness and contrast adjustments to the L channel
    l = cv2.addWeighted(l, contrast, np.zeros_like(l), 0, brightness - 128)

    # Apply detail level adjustment to the L channel
    l = cv2.GaussianBlur(l, (0, 0), sigmaX=(1 - detail_level) * 20 + 1)

    # Merge channels and convert back to BGR color space
```

```
    lab = cv2.merge((l, a, b))
    out = cv2.cvtColor(lab, cv2.COLOR_LAB2BGR)

    return out

def auto_correlation_matrix(image, window_size=3):

    # Compute the derivatives of the image using Sobel kernels
    Ix = cv2.Sobel(image, cv2.CV_32F, 1, 0, ksize=3)
    Iy = cv2.Sobel(image, cv2.CV_32F, 0, 1, ksize=3)

    # Compute the elements of the structure tensor
    auto_rr_matrix_xx = cv2.GaussianBlur(Ix**2, (window_size, window_size), 0)
    auto_rr_matrix_yy = cv2.GaussianBlur(Iy**2, (window_size, window_size), 0)
    auto_rr_matrix_xy = cv2.GaussianBlur(Ix*Iy, (window_size, window_size), 0)

    auto_rr_matrix = [auto_rr_matrix_xx, auto_rr_matrix_yy, auto_rr_matrix_xy]

    return auto_rr_matrix


def gaussian_smooth(image, sigma=2):
    # Create a 1D Gaussian kernel
    kernel_size = 2 * round(3 * sigma) + 1
    kernel = cv2.getGaussianKernel(kernel_size, sigma)

    # Compute the 2D Gaussian kernel
    kernel = np.outer(kernel, kernel.transpose())

    # Normalize the kernel to sum to 1
    kernel /= np.sum(kernel)

    # Apply the Gaussian filter to the input image
    smoothed_image = cv2.filter2D(image, -1, kernel)

    return smoothed_image


def computeFeatureMeasure(I, k=0.04, window_size=3):

    auto_rr_matrix = auto_correlation_matrix(I, window_size)
    [xx, yy, xy] = auto_rr_matrix
    xx = gaussian_smooth(xx)
    yy = gaussian_smooth(yy)
    xy = gaussian_smooth(xy)
    # Compute the Harris response
    det = xx * yy - xy ** 2
```

```
    trace = xx + yy
    R = (det+1)/ (trace+1)
    R /= np.mean(R)
    #print(np.max(abs(R)))
    return R

def FeatureMeasure2Points(R, npoints,threshold = 0.01):

    # Find the locations of local maxima above the threshold
    h, w = R.shape
    mask = np.zeros((h, w), np.uint8)
    mask[R > threshold] = 1

    # Use morphological dilation to find the connected regions
    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
    mask_erosion1 = cv2.erode(mask,          kernel,iterations = 2)
    mask_dilate1 = cv2.dilate(mask_erosion1, kernel,iterations = 2)
    mask_erosion2 = cv2.erode(mask_dilate1,  kernel,iterations = 5)
    mask_dilate2 = cv2.dilate(mask_erosion2, kernel,iterations = 2)
    mask_erosion3 = cv2.erode(mask_dilate2,  kernel,iterations = 5)
    mask_dilate3 = cv2.dilate(mask_erosion3, kernel,iterations = 2)
    mask_erosion4 = cv2.erode(mask_dilate3,  kernel,iterations = 5)
    mask_dilate4 = cv2.dilate(mask_erosion4, kernel,iterations = 2)

    #cv2.imshow('imgh',mask_dilate1*250)
    # Compute the centroid of each connected region as the feature point␣
 ↪location
    num_labels, labels, stats1, centroids1 = cv2.
 ↪connectedComponentsWithStats(mask_dilate1)
    num_labels, labels, stats2, centroids2 = cv2.
 ↪connectedComponentsWithStats(mask_dilate2)
    num_labels, labels, stats3, centroids3 = cv2.
 ↪connectedComponentsWithStats(mask_dilate3)
    num_labels, labels, stats4, centroids4 = cv2.
 ↪connectedComponentsWithStats(mask_dilate4)

    # Remove the background label
    centroids_=np.concatenate((centroids1[1:],centroids2[1:],centroids3[1:
 ↪],centroids4[1:]), axis=0)
    stats_=np.concatenate((stats1[1:],stats2[1:],stats3[1:],stats4[1:]), axis=0)
    stats_=stats_[:,4]
    # Sample a fixed number of feature points from the list of all feature␣
 ↪points
    centroids=[]
    for idx in range(len(stats_)):
        if 100 < stats_[idx] < 500:
```

```
            centroids=np.concatenate((centroids, centroids_[idx]), axis=0)
    centroids=np.reshape(centroids, (int(len(centroids)/2),2))

    if len(centroids) > npoints:
        centroids = centroids[:npoints]

    # Return the x, y coordinates of the feature points and the mask of the␣
 ↪detected regions
    x = centroids[:, 0]
    y = centroids[:, 1]
    mask = np.zeros_like(R)
    for point in centroids:
        mask[int(point[1]), int(point[0])] = 1

    return x, y, mask

def extract_bordered_patch(img, x, y, w, h, border):

    bordered_img = cv2.copyMakeBorder(img, border, border, border, border, cv2.
 ↪BORDER_CONSTANT)

    # Adjust the patch coordinates to account for the border
    x += border
    y += border

    # Extract the patch from the bordered image
    patch = bordered_img[int(y-h//2):int(y+h//2)+1,int( x-w//2):int(x+w//2)+1]

    # Handle the case where the patch exceeds the boundary
    patch_height, patch_width = patch.shape[:2]
    if patch_height != h or patch_width != w:
        # The patch exceeds the boundary
        # Crop the patch to the available size
        patch = patch[:h+1, :w+1]

    return patch


def generateFeatureDescriptors(I, x, y, patch_size=21):
    # Define a square patch around each feature point
    half_patch_size = patch_size // 2
    num_points = len(x)
    Dlist = np.zeros((num_points, patch_size ** 2), dtype=np.float32)
    for i in range(num_points):
        patch=extract_bordered_patch(I, x[i], y[i], patch_size,patch_size ,␣
 ↪border=patch_size)
        norm_patch = (patch - np.mean(patch)) / np.std(patch)
```

```
        norm_kernel = np.ones(norm_patch.shape) / norm_patch.size
        inc_patch = cv2.filter2D(norm_patch, -1, norm_kernel)
        inc_descriptor=inc_patch.flatten()
        inc_descriptor /= np.linalg.norm(inc_descriptor)
        Dlist[i, :] = inc_descriptor

    return Dlist


def show_descriptor_image(image, x, y, Dlist, patch_size=21):
    # Draw a square patch around each feature point and display the image
    half_patch_size = patch_size // 2
    num_points = len(x)
    for i in range(num_points):
        cv2.rectangle(image, (int(x[i]-half_patch_size),␣
 ↪int(y[i]-half_patch_size)),
                      (int(x[i]+half_patch_size), int(y[i]+half_patch_size)),␣
 ↪marker_color, 1)
    #cv2.imshow('Patch', image)
    #cv2.waitKey(0)
    #cv2.destroyAllWindows()

    return image


def transform_point(H1to2, x, y):
    point1 = np.array([x, y, 1])
    point2 = np.dot(H1to2, point1)
    point2_cartesian = point2 / point2[2]
    u, v = point2_cartesian[:2]

    return u, v


def eval_TFPN_s(kp1, kp2, selected_matches, all_matches, trans_matrix,␣
 ↪judge_threshhold=16):
    all_num = max(len(kp1), len(kp2))
    selected_num = len(selected_matches)
    P, N, TP, FN, FP, TN = [0, 0, 0, 0, 0, 0]

    for single_match in all_matches:
        try:
            single_match = single_match[0]
        except:
            single_match = single_match
        pos_kp1 = kp1[single_match.queryIdx].pt
        pos_kp1_ = transform_point(trans_matrix, pos_kp1[0], pos_kp1[1])
```

```
        pos_kp2 = kp2[single_match.trainIdx].pt
        distance = np.linalg.norm(np.array(pos_kp2)-np.array(pos_kp1_))  # type:
 ↪ ignore
        if distance < judge_threshhold:
            P += 1
    N=all_num-P

    TP_matches=[]
    for single_match in selected_matches:
        try:
            single_match = single_match[0]
        except:
            single_match = single_match
        pos_kp1 = kp1[single_match.queryIdx].pt
        pos_kp1_ = transform_point(trans_matrix, pos_kp1[0], pos_kp1[1])
        pos_kp2 = kp2[single_match.trainIdx].pt
        distance = np.linalg.norm(np.array(pos_kp2)-np.array(pos_kp1_))  # type:
 ↪ ignore
        single_match=[pos_kp1,pos_kp2]
        if distance < judge_threshhold:
             TP+= 1
             TP_matches.append(single_match)

    FP=selected_num-TP
    FN = P-TP
    TN = N-FP
    TPR = TP/(max((TP+FN), 1))
    FPR = FP/(max((FP+TN), 1))
    PPV = TP/(max((TP+FP), 1))
    ACC = (TP+TN)/(P+N)

    result_array_d=[P, N, TP, FN, FP, TN, TPR, FPR, PPV, ACC]
    return result_array_d, TP_matches

def eval_TFPN(kp1_x, kp1_y, kp2_x, kp2_y,Dist,Matchlist, trans_matrix,␣
 ↪judge_threshhold=16):
    all_num = kp1_x.size*kp2_x.size
    selected_num = len(Matchlist)
    P, N, TP, FN, FP, TN = [0, 0, 0, 0, 0, 0]

    for i in range(Dist.shape[0]):
        for j in range(Dist.shape[1]):
            pos_kp1=[kp1_x[i],kp1_y[i]]
            pos_kp1_ = transform_point(trans_matrix, pos_kp1[0], pos_kp1[1])
            pos_kp2=[kp2_x[j],kp2_y[j]]
            distance = np.linalg.norm(np.array(pos_kp2)-np.array(pos_kp1_)) #␣
 ↪type: ignore
```

```
            if distance < judge_threshhold:
                P += 1
    P=int(P/2)
    N=all_num-P

    TP_matches=[]
    for queryIdx,trainIdx in Matchlist:
        pos_kp1=[kp1_x[queryIdx],kp1_y[queryIdx]]
        pos_kp1_ = transform_point(trans_matrix, pos_kp1[0], pos_kp1[1])
        pos_kp2=[kp2_x[trainIdx],kp2_y[trainIdx]]
        distance = np.linalg.norm(np.array(pos_kp2)-np.array(pos_kp1_))  # type:
 ↪ ignore
        if distance < judge_threshhold:
            TP += 1
            ␣
 ↪single_match=[[kp1_x[queryIdx],kp1_y[queryIdx]],[kp2_x[trainIdx],kp2_y[trainIdx]]]
            TP_matches.append(single_match)

    TP=int(TP/2)
    FP=selected_num-TP
    FN = P-TP
    TN = N-FP
    TPR = TP/(max((TP+FN), 1))
    FPR = FP/(max((FP+TN), 1))
    PPV = TP/(max((TP+FP), 1))
    ACC = (TP+TN)/(P+N)

    result_array_d=[P, N, TP, FN, FP, TN, TPR, FPR, PPV, ACC]
    return result_array_d, TP_matches


def print_Rarray(arrayname, number_array):
    str_array = arrayname+': '
    variable_names = ['P', 'N', 'TP', 'FN','FP', 'TN', 'TPR', 'FPR', 'PPV',␣
 ↪'ACC']
    for i in range(0, 6):
        str_array += variable_names[i]+': '+str(number_array[i])+', '
    for i in range(6, 10):
        str_array += variable_names[i]+': ' + \
            '{:.3f}'.format(number_array[i])+', '
    str_array += "\n"
    return str_array


def DetectAndCompute(image, max_points):

    sift = cv2.SIFT_create()
```

```
    keypoints = sift.detect(image)
    keypoints = sorted(keypoints, key=lambda x: x.response, reverse=True)
    keypoints = keypoints[:max_points]

    keypoints, descriptors = sift.compute(image, keypoints)
    descriptors = descriptors[:max_points]

    return keypoints, descriptors

def evaluate_INC(descriptor1, descriptor2):

    eps = 1e-6 # A small number to prevent division by zero
    mag1 = np.linalg.norm(descriptor1) + eps
    mag2 = np.linalg.norm(descriptor2) + eps
    dot_product = np.dot(descriptor1, descriptor2)
    inc = 1.0 - dot_product / (mag1 * mag2)
    if dot_product>1:
        dot_product=dot_product
    return inc

def computeDescriptorDistances(Dlist1, Dlist2):
    Dist = np.zeros((Dlist1.shape[0], Dlist2.shape[0]))
    for i in range(Dlist1.shape[0]):
        for j in range(Dlist2.shape[0]):
            #dist[i, j] = np.sqrt(np.sum((Dlist1[i] - Dlist2[j])**2))
            Dist[i, j] = evaluate_INC(Dlist1[i], Dlist2[j])

    return Dist


def computeDescriptorMatchs_s(Dlist1, Dlist2):
    sift = cv2.SIFT_create()
    matcher = cv2.DescriptorMatcher_create(cv2.DescriptorMatcher_BRUTEFORCE)
    Matchs = matcher.knnMatch(Dlist1, Dlist2, k=2)
    return Matchs

def draw_correspondences(image1, image2, points,color1,color2):

    # Convert points to integers
    points = np.round(points).astype(int)

    # Draw points on images
    radius = 4
    thickness = -1
    for p in points:
        image1 = cv2.circle(image1, tuple(p[0]), radius, color1, thickness)
        image2 = cv2.circle(image2, tuple(p[1]), radius, color1, thickness)
```

```
    # Draw line connecting points

    thickness = 1
    image = np.concatenate((image1, image2), axis=1)
    for p in points:
        image = cv2.line(image, tuple(p[0]), tuple(p[1] + [image1.shape[1],␣
 ↪0]), color2, thickness)

    return image

def post_Distance2Matches_s(MatchList, All_MatchList, kp1, kp2, img_o, img_t,␣
 ↪trans_matrix):

    result_array_d, TP_matches = eval_TFPN_s(kp1, kp2, MatchList,␣
 ↪All_MatchList, trans_matrix)

    img_matches = cv2.drawMatches(img_o, kp1, img_t, kp2, TP_matches, None, ␣
 ↪flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
                                                                    ␣
 ↪#matchColor=marker_color,
    return result_array_d, img_matches


def Distance2Matches_DistThresh(Dist, Th1=100):
    # Distance threshhold
    # Apply distance thresholding and select good matches
    MatchList1 = []
    for i in range(Dist.shape[0]):
        for j in range(Dist.shape[1]):
            if Dist[i, j] < Th1:
                MatchList1.append([i, j])

    return MatchList1


def Distance2Matches_NearestMatch(Dist, Th2=0.6):
    # Nearest match
    # Sort matches by distance and select top x
    Dlist = []
    for i in range(Dist.shape[0]):
        for j in range(Dist.shape[1]):
            Dlist.append([Dist[i, j], i, j])
    Dlist = sorted(Dlist, key=lambda x: x[0])
    MatchList2_ = Dlist[:int(len(Dlist)*Th2/len(Dist))]

    MatchList2 = []
    for m in MatchList2_:
```

```
        MatchList2.append(m[1:3])

    return MatchList2


def Distance2Matches_NearestRatio(Dist, Th3=0.7):
    # NNDR

    DD = []
    for i in range(Dist.shape[0]):
        Dlist = []
        for j in range(Dist.shape[1]):
            Dlist.append([Dist[i, j], i, j])
        Dlist = sorted(Dlist, key=lambda x: x[0])
        DD.append(Dlist[:2])

    MatchList3 = []
    for DDD in DD:
        if DDD[0][0] < Th3 * DDD[1][0]:
            MatchList3.append(DDD[0][1:3])

    return MatchList3


def Distance2Matches_DistThresh_s(AllMatchList, threshold_match_d=100):
    # Distance threshhold
    # Apply distance thresholding and select good matches
    MatchList1 = []
    for m in AllMatchList:
        if m[0].distance < threshold_match_d:
            MatchList1.append(m[0])

    return MatchList1


def Distance2Matches_NearestMatch_s(AllMatchList, threshold_match_n=0.6):
    # Nearest match
    # Sort matches by distance and select top x
    AllMatchList = sorted(AllMatchList, key=lambda x: x[0].distance)
    MatchList2_ = AllMatchList[:int(len(AllMatchList)*threshold_match_n)]
    MatchList2 = []
    for m in MatchList2_:
        MatchList2.append(m[0])

    return MatchList2
```

```
def Distance2Matches_NearestRatio_s(AllMatchList, threshold_match_nn=0.7):
    # NNDR
    MatchList3 = []
    for m, n in AllMatchList:
        if m.distance < threshold_match_nn * n.distance:
            MatchList3.append(m)

    return MatchList3
```

[ ]:
```
# Load images
img_o = read_ppm_file(folder+image_id1)
img_o2 = read_ppm_file(folder+image_id2)
img_o3 = read_ppm_file(folder+image_id3)

gray_o_ = cv2.cvtColor(img_o, cv2.COLOR_BGR2GRAY)
gray_o3_ = cv2.cvtColor(img_o3, cv2.COLOR_BGR2GRAY)
# Load homography matrices
H1to2p = np.loadtxt(folder+'H1to2p'); H1to3p = np.loadtxt(folder+'H1to3p');␣
 ↪H1to4p = np.loadtxt(folder+'H1to4p'); H1to5p = np.loadtxt(folder+'H1to5p');␣
 ↪H1to6p = np.loadtxt(folder+'H1to6p')

HH_loop = [(H1to2p, 'H2p_'), (H1to3p, 'H3p_'), (H1to4p, 'H4p_'), (H1to5p,␣
 ↪'H5p_'), (H1to6p, 'H6p_')]
for TMatrix, Hid in HH_loop:
    # Apply transformation matrix to image
    sub_folder = folder+Hid
    print(sub_folder)
    img_t = cv2.warpPerspective(img_o2, TMatrix, (img_o2.shape[1], img_o2.
 ↪shape[0]))
    gray_t_ = cv2.cvtColor(img_t, cv2.COLOR_BGR2GRAY)
    gray_o = unify_contrast_brightness(gray_o_)
    gray_o3 = unify_contrast_brightness(gray_o3_)
    gray_t = unify_contrast_brightness(gray_t_)

    # Display original and transformed images
    cv2.imwrite(sub_folder+'0101_Original_image'+'.jpg', gray_o3_)
    cv2.imwrite(sub_folder+'0102_Transformed'+'.jpg', img_t)

    cv2.imwrite(sub_folder+'0201_Original_image gray_o'+'.jpg', gray_o3)
    cv2.imwrite(sub_folder+'0202_Transformed gray_t'+'.jpg', gray_t)

    # personal

    R_img_o = computeFeatureMeasure(gray_o)
    R_img_t = computeFeatureMeasure(gray_t)

    num_points = 800
```

```
  #x_img_o, y_img_o, mask_img_o = FeatureMeasure2Points(R_img_o, num_points,0.
↪5)
  #x_img_t, y_img_t, mask_img_t = FeatureMeasure2Points(R_img_t, num_points,0.
↪5)
  #num_points=min(len(x_img_o),len(x_img_t))
  x_img_o, y_img_o, mask_img_o = FeatureMeasure2Points(R_img_o, num_points,0.
↪8)
  x_img_t, y_img_t, mask_img_t = FeatureMeasure2Points(R_img_t, num_points,0.
↪8)

  img_o_save1 = img_o3.copy();     img_t_save1 = img_t.copy()

  for i in range(len(x_img_o)):
      cv2.circle(img_o_save1, (int(x_img_o[i]), int(y_img_o[i])), 3,␣
↪marker_color, -1)

  for i in range(len(x_img_t)):
      cv2.circle(img_t_save1, (int(x_img_t[i]), int(y_img_t[i])), 3,␣
↪marker_color, -1)

  cv2.imwrite(sub_folder+'0301_Feature_O'+'.jpg', img_o_save1)
  cv2.imwrite(sub_folder+'0302_Feature_T'+'.jpg', img_t_save1)

  patch_size = 21

  Dlist_img_o = generateFeatureDescriptors(gray_o, x_img_o, y_img_o,␣
↪patch_size)
  img_o_save2 = show_descriptor_image(img_o_save1.copy(), x_img_o, y_img_o,␣
↪Dlist_img_o, patch_size)

  Dlist_img_t = generateFeatureDescriptors(gray_t, x_img_t, y_img_t,␣
↪patch_size)
  img_t_save2 = show_descriptor_image(img_t_save1.copy(), x_img_t, y_img_t,␣
↪Dlist_img_t, patch_size)

  cv2.imwrite(sub_folder+'0401_Descriptors_O' + '.jpg', img_o_save2)
  cv2.imwrite(sub_folder+'0402_Descriptors_T' + '.jpg', img_t_save2)

  Dist = computeDescriptorDistances(Dlist_img_o, Dlist_img_t)

  threshold_match_d = 3.5;    threshold_match_n = 10.2;    threshold_match_nn␣
↪= 0.8

  threshold_match_d=np.mean(np.amin(Dist, axis=1))*threshold_match_d
  MatchList1 = Distance2Matches_DistThresh(Dist, threshold_match_d)
  MatchList2 = Distance2Matches_NearestMatch(Dist, threshold_match_n)
```

```
  MatchList3 = Distance2Matches_NearestRatio(Dist, threshold_match_nn)

  result_array_d, TP_matches_d   =eval_TFPN(x_img_o, y_img_o, x_img_t,␣
↪y_img_t, Dist, MatchList1, TMatrix)
  result_array_n, TP_matches_n   =eval_TFPN(x_img_o, y_img_o, x_img_t,␣
↪y_img_t, Dist, MatchList2, TMatrix)
  result_array_nn, TP_matches_nn =eval_TFPN(x_img_o, y_img_o, x_img_t,␣
↪y_img_t, Dist, MatchList3, TMatrix)

  result_image_d  =draw_correspondences(img_o_save1.copy(),img_t_save1.
↪copy(),TP_matches_d, (0,0,255),(0,255,0))
  result_image_n  =draw_correspondences(img_o_save1.copy(),img_t_save1.
↪copy(),TP_matches_n, (0,0,255),(0,255,0))
  result_image_nn =draw_correspondences(img_o_save1.copy(),img_t_save1.
↪copy(),TP_matches_nn,(0,0,255),(0,255,0))

  cv2.imwrite(sub_folder+'0501_Matches_d' + '.jpg', result_image_d)
  cv2.imwrite(sub_folder+'0502_Matches_n' + '.jpg', result_image_n)
  cv2.imwrite(sub_folder+'0503_Matches_nn' + '.jpg', result_image_nn)

  string1 = print_Rarray("FixDist", result_array_d)
  string2 = print_Rarray("Nearest", result_array_n)
  string3 = print_Rarray("N N D R", result_array_nn)
  print(string1+string2+string3)

  # sift
  kp1, Dlist1 = DetectAndCompute(gray_o, 500)
  kp2, Dlist2 = DetectAndCompute(gray_t, 500)

  All_Matchs = computeDescriptorMatchs_s(Dlist1, Dlist2)

  threshold_match_d = 120;    threshold_match_n = 0.6;    threshold_match_nn␣
↪= 0.7
  MatchList1_s = Distance2Matches_DistThresh_s(All_Matchs, threshold_match_d)
  MatchList2_s = Distance2Matches_NearestMatch_s(All_Matchs,␣
↪threshold_match_n)
  MatchList3_s = Distance2Matches_NearestRatio_s(All_Matchs,␣
↪threshold_match_nn)

  result_array_s_d , TP_matches_s_d  = eval_TFPN_s(kp1, kp2, MatchList1_s,␣
↪All_Matchs, TMatrix)
  result_array_s_n , TP_matches_s_n  = eval_TFPN_s(kp1, kp2, MatchList2_s,␣
↪All_Matchs, TMatrix)
  result_array_s_nn, TP_matches_s_nn = eval_TFPN_s(kp1, kp2, MatchList3_s,␣
↪All_Matchs, TMatrix)
```

```
   result_image_s_d  =draw_correspondences(img_o3.copy(),img_t.
↪copy(),TP_matches_s_d, (0,255,0),(0,0,255))
   result_image_s_n  =draw_correspondences(img_o3.copy(),img_t.
↪copy(),TP_matches_s_n, (0,255,0),(0,0,255))
   result_image_s_nn =draw_correspondences(img_o3.copy(),img_t.
↪copy(),TP_matches_s_nn,(0,255,0),(0,0,255))

   cv2.imwrite(sub_folder+'0601_Sift_Matches_d'+'.jpg', result_image_s_d)
   cv2.imwrite(sub_folder+'0602_Sift_Matches_n'+'.jpg', result_image_s_n)
   cv2.imwrite(sub_folder+'0603_Sift_Matches_nn'+'.jpg', result_image_s_nn)

   string1 = print_Rarray("FixDist_s", result_array_s_d)
   string2 = print_Rarray("Nearest_s", result_array_s_n)
   string3 = print_Rarray("N N D R_s", result_array_s_nn)
   print(string1+string2+string3)


  ␣
↪Result=[result_array_d,result_array_n,result_array_nn,result_array_s_d,result_array_s_n,res
   df = pd.DataFrame(Result,columns=['P', 'N', 'TP', 'FN', 'FP', 'TN', 'TPR',␣
↪'FPR', 'PPV', 'ACC'],
                     index=[ 'FixDist', 'Nearest', 'N N D R', 'FixDistS',␣
↪'NearestS', 'N N D RS'])

   df.to_excel(sub_folder+'result.xlsx')
```