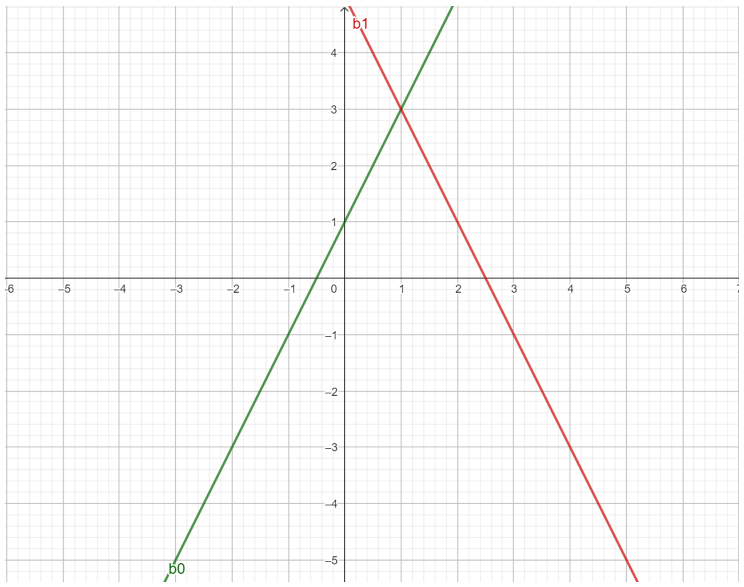
DATA.ML.300 Computer Vision Exercise 4

Uyen Phan

01 2024

1

a) We have: $y = mx + b \Rightarrow b = -mx + y$
With two points:
$(x_0, y_0) = (-2, 1) \Rightarrow b_0 = 2m_0 + 1$
$(x_1, y_1) = (2, 5) \Rightarrow b_0 = -2m_0 + 5.$
Plot line b_0 and b_1 onto the parameter plane (m, b):



It can be seen from the figure that the two lines intersect at $(m',\ b') = (1, 3)$

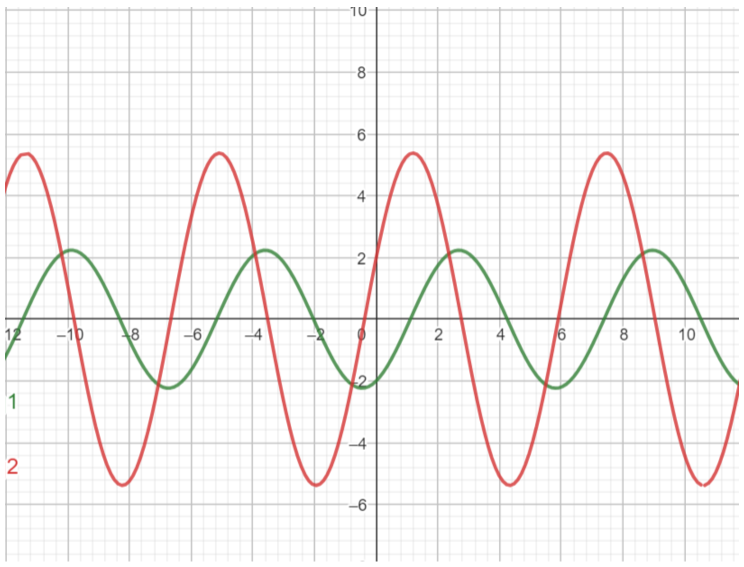
b) Each point can be represented in polar form:
$-2cos\theta + sin\theta = \rho$
$2cos\theta + 5sin\theta = \rho$

Or: $-2cos\theta + sin\theta = 2cos\theta + 5sin\theta \Rightarrow cos\theta + sin\theta = 0 \Rightarrow \theta = -\frac{\pi}{4} + k\pi$

Since $-\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2}$, $\theta = -\frac{\pi}{4}$ when $k = 0$

Find ρ: $-2cos(-\frac{\pi}{4}) + sin(-\frac{\pi}{4}) = -\frac{3\sqrt{2}}{2}$

The point of intersection $(\theta',\ \rho') = (-\frac{\pi}{4}, -\frac{3\sqrt{2}}{2})$ can be seen from the plot below

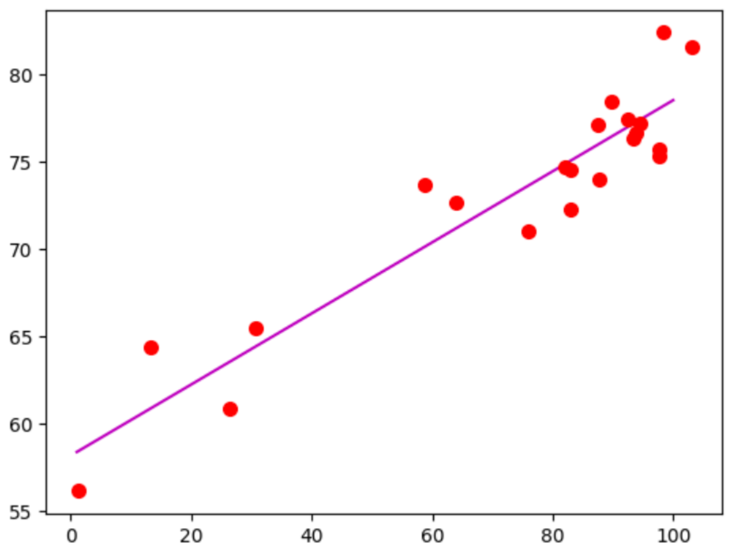


2

Implementation code:

```
while N > sample_count:
    samples = np.random.choice(np.arange(len(x)), 2, replace=False)
    id1 = samples[0]  # sample id 1
    id2 = samples[1]  # sample id 2

    ##-your-code-starts-here-##
    l = np.cross(points_h[:, id1], points_h[:, id2])
    l = l / np.sqrt(l[0]**2 + l[1]**2)
    # l = l / np.linalg.norm(l[0:2])
    ##-your-code-ends-here-##

    ##-your-code-starts-here-##
    inliers = np.abs(np.dot(l, points_h)) < t  # binary array
    ##-your-code-ends-here-##
    inlier_count = np.size(inliers)
    if inlier_count > max_inliers:
        best_line = l
        max_inliers = inlier_count

    e = 1 - inlier_count / m
    N = np.log(1 - p) / np.log(1 - (1 - e) ** s)
    sample_count += 1

x_inliers = x[inliers]
y_inliers = y[inliers]
```

Plot for task 2:

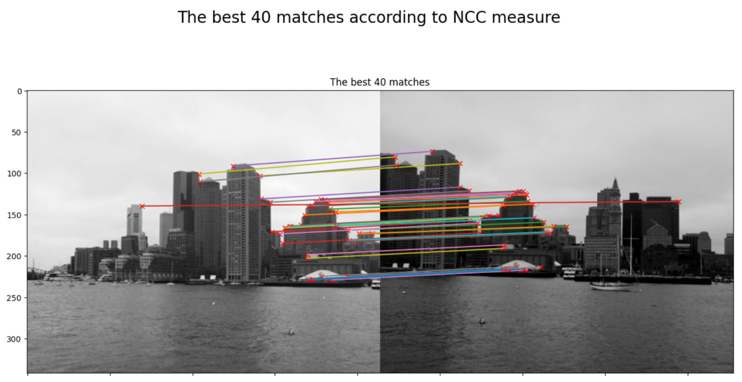


3

Implementation code:

```
nccmat = np.zeros((npts1, npts2))
for i1 in range(npts1):
    for i2 in range(npts2):
        patch1 = patches1[:,:,i1]
        patch2 = patches2[:,:,i2]

        # Compute mean intensity values of patches
        mean1 = np.mean(patch1)
        mean2 = np.mean(patch2)

        # Compute NCC
        numerator = np.sum((patch1 - mean1) * (patch2 - mean2))
        denominator = np.sqrt(np.sum((patch1 - mean1)**2) * np.sum((patch2 - mean2)**2))
        nccmat[i1, i2] = numerator / denominator if denominator != 0 else 0

# Next, compute pairs of patches that are mutually nearest neighbors
# according to the NCC measure
ncc1 = np.amax(nccmat, axis=1)
idn1 = np.argmax(nccmat, axis=1)
ncc2 = np.amax(nccmat, axis=0)
idn2 = np.argmax(nccmat, axis=0)

pairs = []
for k in range(npts1):
    if k == idn2[idn1[k]]:
        pairs.append(np.array([k, idn1[k], ncc1[k]]))
pairs = np.array(pairs)

# We sort the mutually nearest neighbors based on the NCC
sorted_ncc = np.sort(pairs[:,2], axis=0)[::-1]  # Sorting in descending order
id_ncc = np.argsort(pairs[:,2], axis=0)[::-1]
```

Plot for task 3:



NCC performs better in this case because NCC considers the mean and standard deviation of pixel intensities in the patches. This is useful when patches have similar structures but different brightness like this case.

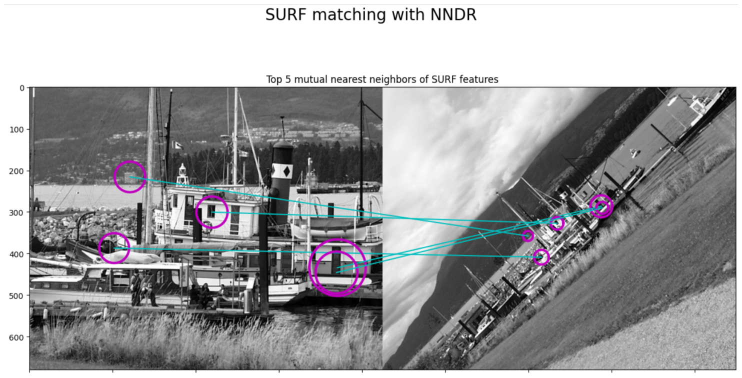
4

Implementation code:

```
distmat_sorted = np.sort(distmat, axis=1)  # each row sorted in ascending order
nndr=np.zeros(pairs.shape[0])  # pre-allocate memory

##-your-code-starts-here-##
for k in range(pairs.shape[0]):
    idx1 = int(pairs[k, 0])
    dist1 = distmat_sorted[idx1, 0]  # nearest distance
    dist2 = distmat_sorted[idx1, 1]  # second nearest distance
    nndr[k] = dist1 / dist2

id_nndr = np.argsort(nndr)
##-your-code-ends-here-##
# How many of the top 5 matches appear to be correct correspondences?
# Five
```

In the top 5 matches, there are only 2 correct correspondences in the case where ordering is based on nearest neighbor distance, while there are 5 correct matches based on NNDR.

Plot for task 4:



SURF is scale and rotation-invariant while Harris corners, on the other hand, are not inherently scale-invariant.

Cases Harris corners may still be better than SURF might be when computational resources are limited, Harris corners might be preferred due to their simplicity; or in applications where corner detection is of primary concern, Harris corners might be sufficient.