Assignment 5

425

Robert MacEachern

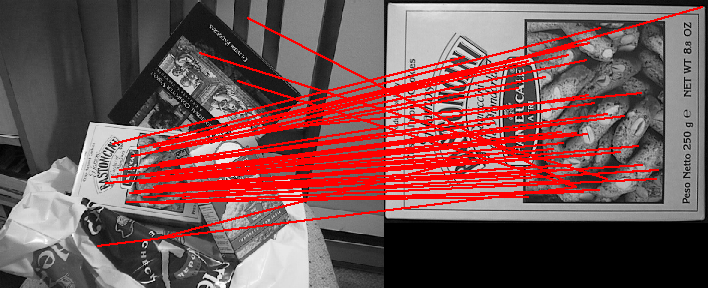
42415091

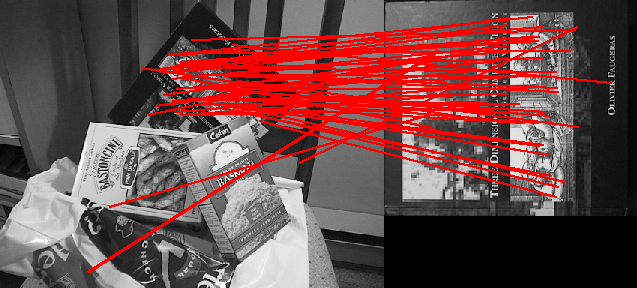
m9c7

Q2:

Print the box image showing the set of matches to the scene image for your suggested threshold value. Write a short description of the particular threshold value used, why you chose it and how important it was to get the value correct.

A2:



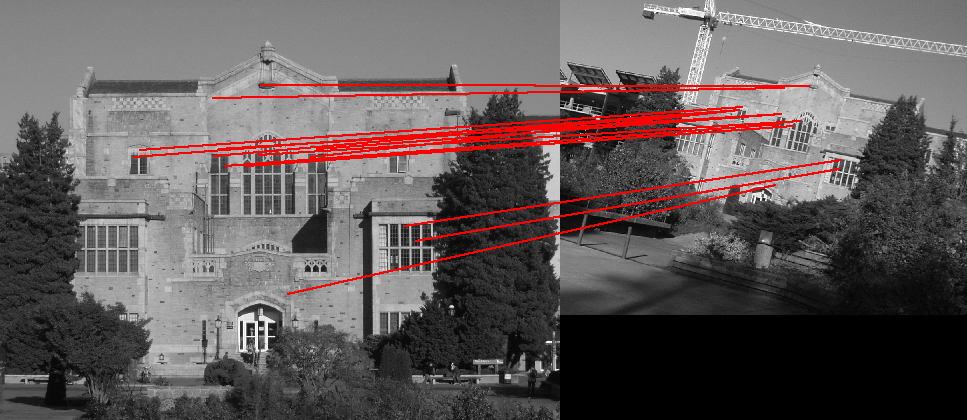




I used the threshold value of 0.75 and I chose it by looking for substantial differences in the angles of matching lines, and then adjusting it until I judged there to be around 10 lines (or slightly fewer) that had a noticiably different angle. It was important to get a correct value for this threshold because (a) if we chose a threshold that was too high, we would risk eliminating correct matches (false negatives) and (b) if we chose a threshold that was too low we would get too many invalid matches (false positives) and our next step (RANSAC) will be much less likely to provide a “correct” model fitting without a large number of samples. For example, if we have 10 obvious outliers there is a chance that RANSAC will use each of those ten as sample points to construct support sets which will clearly lead us to undesirable results. The probability of such undesirable results increases as our threshold drops.

Q2:

Print the resulting library images showing your best set of matches. Write a paragraph summarizing the effects of consistency checking and the degree to which it allowed you to raise the matching threshold.



Angle agreement limit: +/- 20 degrees

Scale agreement limit : 90%

Support set size: 15

Consistentcy checking allows us to compare the orientation and scale differences between matches, which gives us a very effective way of measuring the compatibility between the matches we find. RANSAC allows us to increase the matching threshold because it will (with high probability) eliminate any matches that we found in the previous step that are not in agreement with the largest support set we find. The end result is a set of matches that are in strong agreement with respect to orientation, scale, and for the most part, position. This gives us the opportunity to be slightly less restrictive during the initial filtering (Q1) but still retain a high degree of accuracy.

Script: SIFTmatch.py

from PIL import Image, ImageDraw

from pprint import pprint

import numpy as np

import csv

import math

def ReadKeys(image):

"""Input an image and its associated SIFT keypoints.

The argument image is the image file name (without an extension).

The image is read from the PGM format file image.pgm and the

keypoints are read from the file image.key.

ReadKeys returns the following 3 arguments:

image: the image (in PIL 'RGB' format)

keypoints: K-by-4 array, in which each row has the 4 values specifying

a keypoint (row, column, scale, orientation). The orientation

is in the range [-PI, PI] radians.

descriptors: a K-by-128 array, where each row gives a descriptor

for one of the K keypoints. The descriptor is a 1D array of 128

values with unit length.

"""

im = Image.open(image+'.pgm').convert('RGB')

keypoints = []

descriptors = []

first = True

with open(image+'.key','rb') as f:

reader = csv.reader(f, delimiter=' ', quoting=csv.QUOTE\_NONNUMERIC,skipinitialspace = True)

descriptor = []

for row in reader:

if len(row) == 2:

assert first, "Invalid keypoint file header."

assert row[1] == 128, "Invalid keypoint descriptor length in header (should be 128)."

count = row[0]

first = False

if len(row) == 4:

keypoints.append(np.array(row))

if len(row) == 20:

descriptor += row

if len(row) == 8:

descriptor += row

assert len(descriptor) == 128, "Keypoint descriptor length invalid (should be 128)."

#normalize the key to unit length

descriptor = np.array(descriptor)

descriptor = descriptor / math.sqrt(np.sum(np.power(descriptor,2)))

descriptors.append(descriptor)

descriptor = []

assert len(keypoints) == count, "Incorrect total number of keypoints read."

print "Number of keypoints read:", int(count)

return [im,keypoints,descriptors]

def AppendImages(im1, im2):

"""Create a new image that appends two images side-by-side.

The arguments, im1 and im2, are PIL images of type RGB

"""

im1cols, im1rows = im1.size

im2cols, im2rows = im2.size

im3 = Image.new('RGB', (im1cols+im2cols, max(im1rows,im2rows)))

im3.paste(im1,(0,0))

im3.paste(im2,(im1cols,0))

return im3

def DisplayMatches(im1, im2, matched\_pairs):

"""Display matches on a new image with the two input images placed side by side.

Arguments:

im1 1st image (in PIL 'RGB' format)

im2 2nd image (in PIL 'RGB' format)

matched\_pairs list of matching keypoints, im1 to im2

Displays and returns a newly created image (in PIL 'RGB' format)

"""

im3 = AppendImages(im1,im2)

offset = im1.size[0]

draw = ImageDraw.Draw(im3)

for match in matched\_pairs:

draw.line((match[0][1], match[0][0], offset+match[1][1], match[1][0]),fill="red",width=2)

im3.show()

return im3

def absoluteAngleDifference(a, b):

'''

Returns the absolute difference (in degrees) between two angles (specified

in radians)

a - an angle specified in radians

b - an angle specified in radians

'''

aa = math.degrees(a) + 360

bb = math.degrees(b) + 360

return abs(aa - bb) % 180

def match(image1,image2):

"""Input two images and their associated SIFT keypoints.

Display lines connecting points that meet our experimental thresholds for

SIFT and RANSAC.

The arguments image1 and image2 are file names without file extensions.

Returns the number of matches displayed.

Example: match('scene','book')

"""

im1, keypoints1, descriptors1 = ReadKeys(image1)

im2, keypoints2, descriptors2 = ReadKeys(image2)

#

# REPLACE THIS CODE WITH YOUR SOLUTION (ASSIGNMENT 5, QUESTION 3)

#

#Generate five random matches (for testing purposes)

angle\_threshold = 0.75

matched\_pairs = []

for i in range(len(descriptors1)):

angles = []

for j in range(len(descriptors2)):

# We compare each descriptor in descriptors1 with

# every descriptor in descriptors2, calculating the

# angle between each pair

dot = np.dot(descriptors1[i], descriptors2[j])

angles.append(math.acos(dot))

# Sort the resulting angles and select the

# two smallest

sortedAngles = sorted(angles)

best = sortedAngles[0]

second = sortedAngles[1]

# We calculate the ratio between the best and second best

# angles, and only count it as a match if the ratio is less

# than our threshold

if (best/second <= angle\_threshold):

bestIndex = angles.index(best)

matched\_pairs.append([keypoints1[i], keypoints2[bestIndex]])

#print 'meets threshold:', best, second

# maximum difference (in degrees) between change in orientation. Used in RANSAC

orientationLimit = 20

# maxiumum scale ratio. Used in RANSAC

scaleLimit = 0.9

largestSupport = []

for k in range(10):

randomMatch = matched\_pairs[np.random.randint(len(matched\_pairs))]

deltaScale1 = abs(randomMatch[0][2] - randomMatch[1][2])

deltaOrientation1 = randomMatch[0][3] - randomMatch[1][3]

currentSupport = []

for match in matched\_pairs:

# check for consistency and add it to currentSupport if consistent

deltaOrientation2 = match[0][3] - match[1][3]

difference = absoluteAngleDifference(deltaOrientation1, deltaOrientation2)

if (difference > orientationLimit):

# Is not consistent, move on to next element

continue

# Check scale

deltaScale2 = abs(match[0][2] - match[1][2])

maxScale = max(deltaScale1, deltaScale2)

minScale = min(deltaScale1, deltaScale2)

if (maxScale \* scaleLimit > minScale):

continue

# if we reach this point, it's consistent

# so add it to our support set

currentSupport.append(match)

# Update our largest support set, if necessary

if (len(currentSupport) > len(largestSupport)):

largestSupport = currentSupport

print "length of largest support set: ", len(largestSupport)

#im3 = DisplayMatches(im1, im2, matched\_pairs)

im3 = DisplayMatches(im1, im2, largestSupport)

im3.save('result.png', 'PNG')

return im3

#Test run...

#match('scene','basmati')

match('library', 'library2')