Homework - 4

Sriram Karthik Badam

October 2, 2012

1. **Harris Corner Detector to extract Interest Points**

   The Harris Corner Detector finds the interest points by calculating the variations of pixel intensities over x and y directions (assuming a simple 2D image plane) and then thresholding the Corner Response which is function of eigen values of the covariance matrix formed from the Intensity derivatives.

   The pixels with Corner Response over a threshold value are our interest points because in a sense there is higher chance of finding a corner at that pixel.

   Algorithm:

   (a) The first step is to calculate the variations along x and y i.e., the x-derivative and the y-derivative (I_x, I_y). We can do this using the Sobel Operator.

   $$I_x(x,y) = I(x-1,y-1) - 2*I(x-1,y) + I(x-1,y+1) - I(x+1,y-1) + 2*I(x+1,y) - I(x+1,y+1)$$

   $$I_y(x,y) = I(x-1,y-1) - 2*I(x,y-1) + I(x+1,y-1) - I(x-1,y+1) + 2*I(x,y+1) - I(x+1,y+1)$$

   (b) Next, For every pixel in the image compute the CoVariance matrix from the values of I_x, I_y at neighboring pixels(i.e., the pixels within a window).

   $$C = \begin{bmatrix} \sum {I_x}^2 & \sum I_x I_y \\ \sum I_x I_y & \sum {I_y}^2 \end{bmatrix}$$

   (c) Calculate the eigen values of the Co-Variance Matrix at each pixel using SVD Decomposition.

   $$C = U \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} V^T$$

   (d) Compute the Corner Response $= \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$ where k is a constant (k = 0.04)

   (e) Get rid of the pixels with Corner Response lower than their surrounding pixels (By applying a window). The final Interest points are the pixels with Corner Response greater than a Threshold value.

2. **Finding Correspondence between Interest points in two different images**

   (a) To find the correspondences, first we have to define a feature value for each interest point. We define the feature descriptor for each interest point as a set of values containing the co-ordinates of the pixel and a matrix consisting of the neighbor pixel intensities at a pixel.

(b) Once we get the feature descriptors at each point, we do a iterative comparision of each interest point in the first image with every interest point in the second image to find its match (an exhaustive search). Ofcourse, the matching is done by computing the Discrepancy measure/score between two interest points and finding the correspondence with the maximum/minimum score (depending on the kind of Algorithm). There are two Algorithms that are used for Feature Matching.

i. Sum of Squared Distances (SSD):
In this method, we compute the score by summing up the square of the difference between each neighboring pixel value. Note that the neighboring pixel values are stored in the feature descriptor.

$$SSD_{score} = \frac{1}{m^2}\sum_{i=1}^{n}(IP_1(i) - IP_2(i))^2$$

where $IP_1(i)$ stands for the Intensity at Interest point(corner) i of image 1, n is the number of interest points and m is the size of the neighbor window used. The lower the value of SSD_{score} the more likely that the interest points are similiar. It can be observed that some features maynot be unique in a picture which means when you try to make correspondence you find multiple interest points giving similiar score. To get rid of such points we put a threshold on the value of the ratio of minimum score and second minimum score (SSD_{ratio}).

ii. Normalized Cross Correlation (NCC):
The formula for computing this score is as follows

$$NCC_{score} = \frac{\sum_{i=1}^{n}(IP_1(i) - mean_1)(IP_2(i) - mean_2)}{\sqrt[2]{\sum_{i=1}^{n}(IP_1(i) - mean_1)^2 \sum_{i=1}^{n}(IP_2(i) - mean_2)^2}}$$

where $IP_1(i)$ stands for the Intensity at Interest point i of image 1 and n is the number of interest points.
The larger the value of correlation the more likely that the interest points are similiar. We perform a threshold on the ratio of the maximum score and second maximum score to get rid of the non-unique interest points.

(c) After computing the correspondence between features in both images, we can show them in our final image by appending both images side by side.

3. Speeded Up Robust Feature(SURF)

In contrast to the Harris corner detector, SURF defines a interest point as the pixel where the determinant of Hessian is maximized at some scale(σ value). SURF works over a pyramid(set) of images smoothed with different scale(σ) values.

$$H(x, y, \sigma) = \begin{bmatrix} \frac{\partial^2 ff(x,y,\sigma)}{\partial x^2} & \frac{\partial^2 ff(x,y,\sigma)}{\partial x \partial y} \\ \frac{\partial^2 ff(x,y,\sigma)}{\partial x \partial y} & \frac{\partial^2 ff(x,y,\sigma)}{\partial y^2} \end{bmatrix}$$

The values of the elements in the Hessian matrix are computed by using discrete approximations of the double derivatives.

Note:

(a) Inorder to make calculations faster SURF uses Integral images.

$$I(x,y) = \sum_{i=0}^{x}\sum_{j=0}^{x} f(i,j)$$

where I is the Integral image and f(i,j) is the pixel value of i,j in original image.

(b) Computing the sum of gray levels in a rectangular segment of the integral image takes exactly 4 pixel lookups and computing the derivative takes 6 pixel lookups.

(c) When the double derivatives in the Hessian Matrix are to be computed for every pixel at every scale(σ), the use of Integral image eases up the computation because the number of pixel values needed are less.

The feature descriptor for SURF interest points is a simple 64x1 vector computed by the following steps:(explained in a concise way)

(a) Calculate the dominant direction at each interest point found at some scale by looking at the d_x and d_y values in a 6σ x 6σ window.

(b) Consider a neighborhood of 20σ x 20σ, around the interest point, oriented in the dominant direction. The neighborhood is divided into 4x4 squares and from these 16 squares, we compute the feature descriptor by applying two Haar descriptors relative to the dominant direction.(Infact the results of the Haar operator are summed up in a 5x5 window)

Once the Feature descriptor is created for each interest point we can find the correspondences by computing the Euclidean distance between the feature descriptors (again an exhaustive search). The interest points with lowest distance are considered to be similiar. To get rid of the non-unique points we threshold the ratio of the minimum and the second minimum in this case too.

4. Parameters Chosen for best Feature Extraction and Matching:

The parameters (especially the Threshold values) have a great role in finding Interest points and correspondences. The parameters shown in the ones which tweaked to get good results in terms of less false positives and more correspondences.

Here is a small description of the parameters

(a) W_s = Harris Corner Detector Window Size

(b) R_{Th} = Threshold on Corner Response

(c) SSD_s = SSD Window Size

(d) NCC_s = NCC Window Size

(e) SSD_{Th} = Threshold on SSD score

(f) SSD_{ratio} = Threshold on ratio of SSD minimum, second minimum Scores

(g) NCC_{Th} = Threshold on the NCC score

(h) NCC_{ratio} = Threshold on ratio of NCC maximum, second maximum scores

(i) H_{Th} = Threshold on Hessian value of feature point in SURF

(j) $SURF_{score}$ = Threshold on the SURF distance value between two feature points

(k) $SURF_{ratio}$ = Threshold on the ratio of minimum score and second minimum sore for SURF feature correspondence

	W_s	R_{Th}	SSD_s	NCC_s	SSD_{Th}	SSD_{ratio}	NCC_{Th}	NCC_{ratio}
building1.jpg	05	7000000000	41	15	1300	0.85	0.75	1.2
building2.jpg	05	30000000000	41	15	1300	0.85	0.75	1.2
tower1.jpg	05	700000000	41	53	1300	0.7	0.76	1.07
tower2.jpg	05	800000000	41	53	1300	0.7	0.76	1.07
stereo.jpg	05	700000000	41	53	1100	0.8	0.8	1.2
stereo1.jpg	05	800000000	41	53	1100	0.8	0.8	1.2
sample-a.jpg	05	7000000000	41	53	1100	0.75	0.78	1.1
sample-b.jpg	05	10000000000	41	53	1100	0.75	0.78	1.1

Table 1: Shows the parameter values in Harris Corner Detector

	H_{Th}	$SURF_{score}$	$SURF_{ratio}$
building1.jpg	2000	0.15	0.9
building2.jpg	2000	0.15	0.9
tower1.jpg	450	0.15	0.8
tower2.jpg	450	0.15	0.8
stereo.jpg	1000	0.2	0.9
stereo1.jpg	1000	0.2	0.9
sample-a.jpg	2000	0.15	0.9
sample-b.jpg	2000	0.15	0.9

Table 2: Shows the parameter values in SURF

5. **Observations**:

   (a) The SURF implementation is much faster in terms of time taken and the Results from the SURF have more (or nearly equal) number of correspondences than the NCC feature Matching for the images used. (but adjusting the thresholds, NCC may give better results).

   (b) The SURF implementation fails with the stereo.jpg, stereo1.jpg input. The Harris Corner Detector detects the tips of the flower petals which gives a really good output than SURF in terms number of correct correspondences.

   (c) Although SSD Algorithm seems to work well with the examples it will give more false correspondences when we consider a image pair with different illumination or some scaling.

   (d) The NCC Algorithm is invariant to Affine Mapping (A general observation).

6. Results:(zoom in for a better view)

   The parameters have been adjusted such that all the correspondences are correct i.e., no false positives (or atmost one false positive).

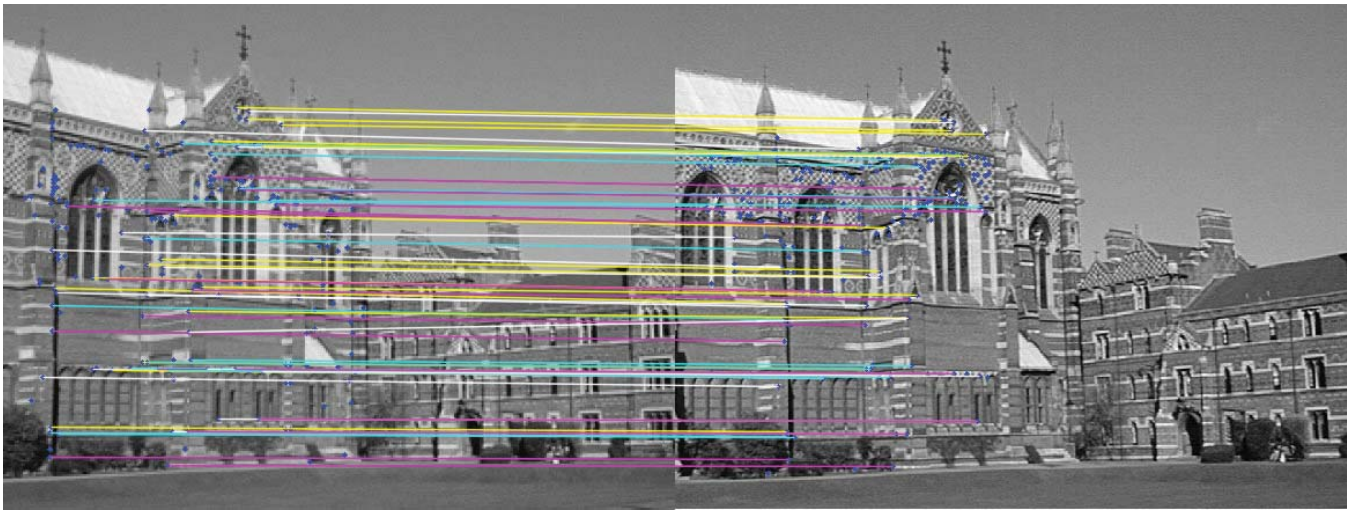

Figure 1: SSD Feature Matching on building1.jpg, building2.jpg - 60 correspondences found

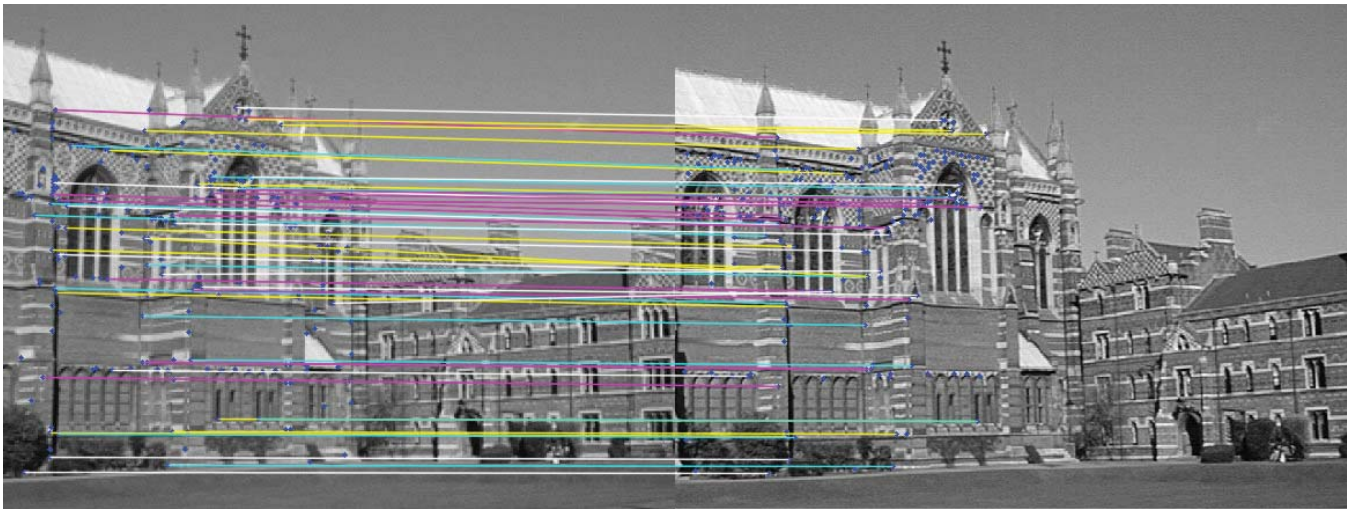

Figure 2: NCC Feature Matching on building1.jpg, building2.jpg - 49 correspondences found

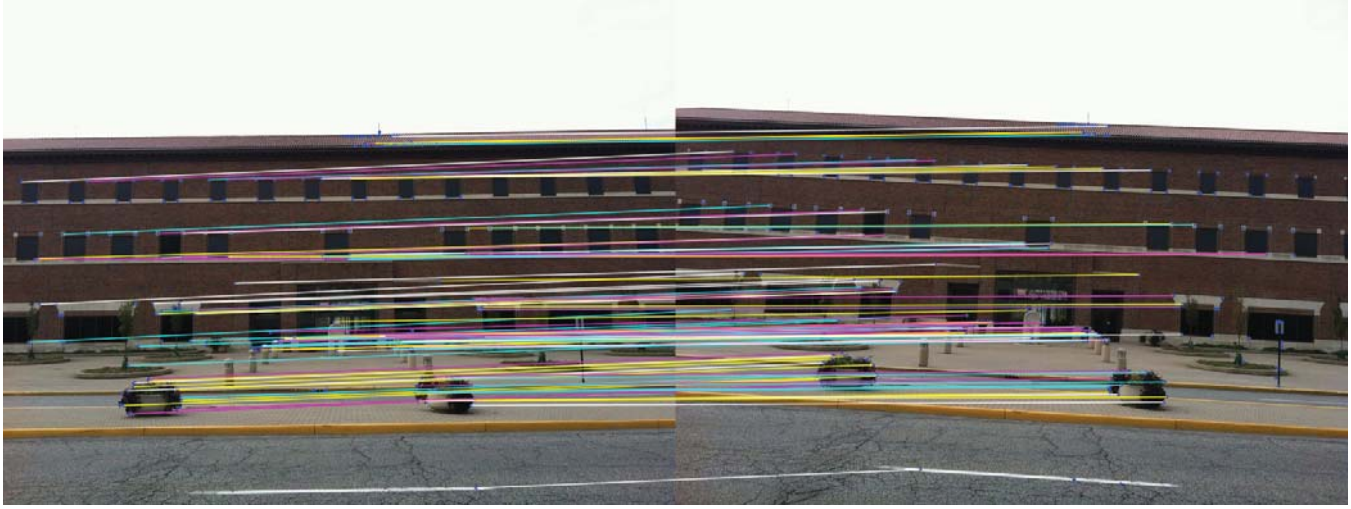

Figure 3: SSD Feature Matching on sample-a.jpg, sample-b.jpg - 98 correspondences found

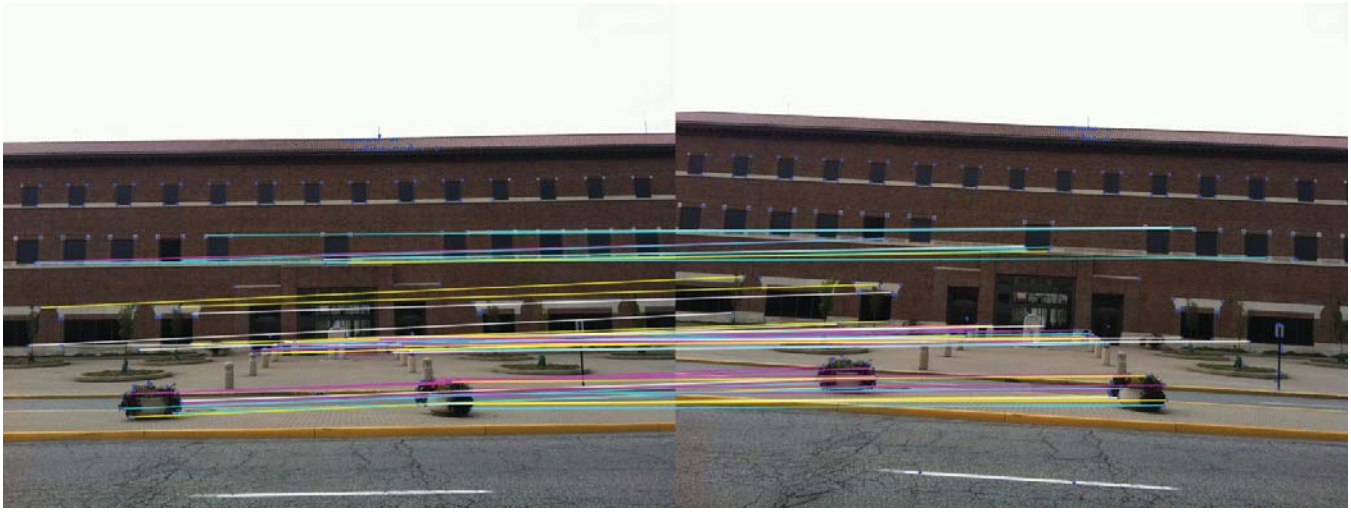

Figure 4: NCC Feature Matching on sample-a.jpg, sample-b.jpg - 58 correspondences found

Figure 5: SSD Feature Matching on stereo.jpg, stereo1.jpg - 21 correspondences found - one false positive

Figure 6: NCC Feature Matching on stereo.jpg, stereo1.jpg - 14 correspondences found

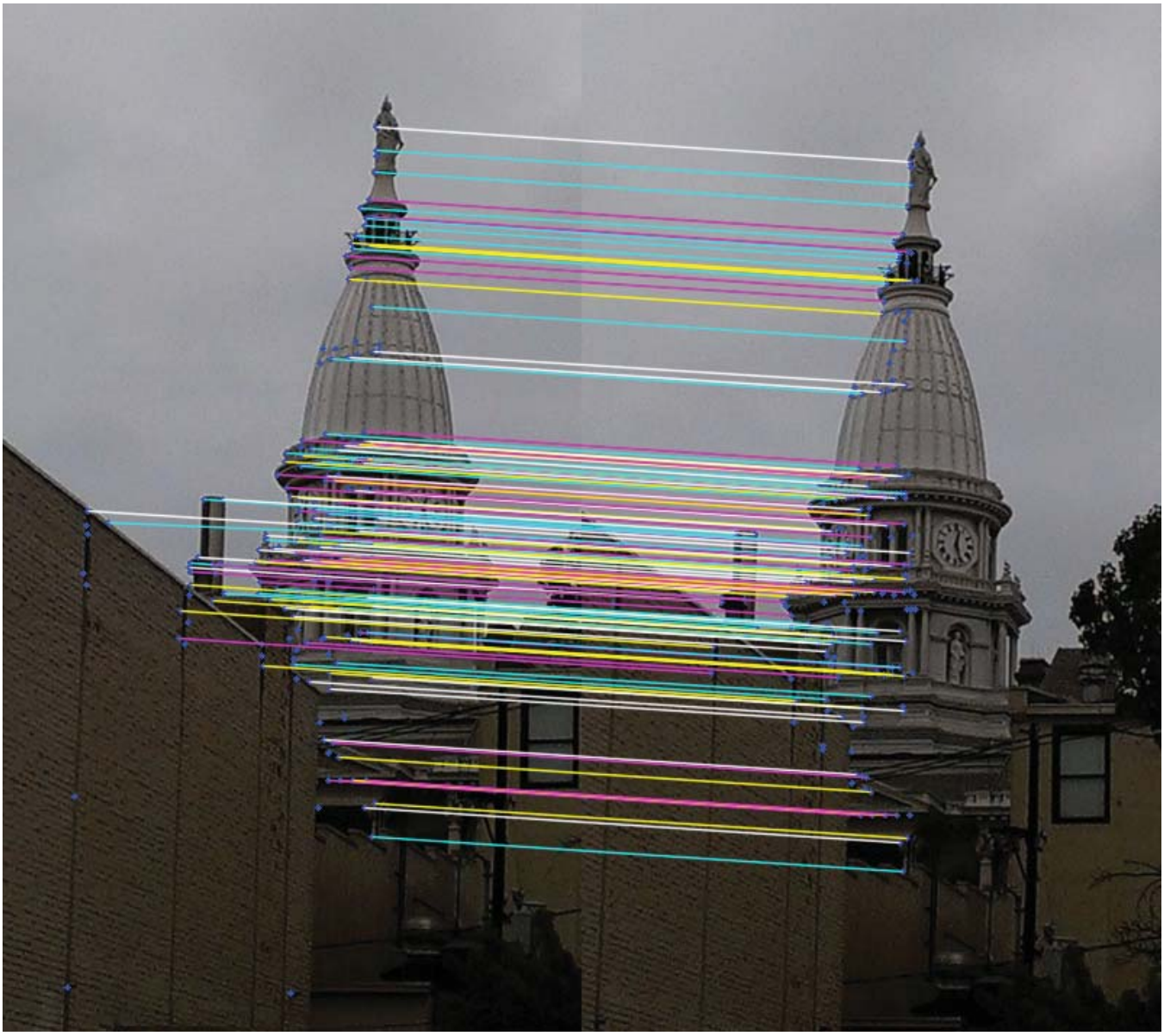

Figure 7: SSD Feature Matching on tower1.jpg, tower2.jpg - 133 correspondences found

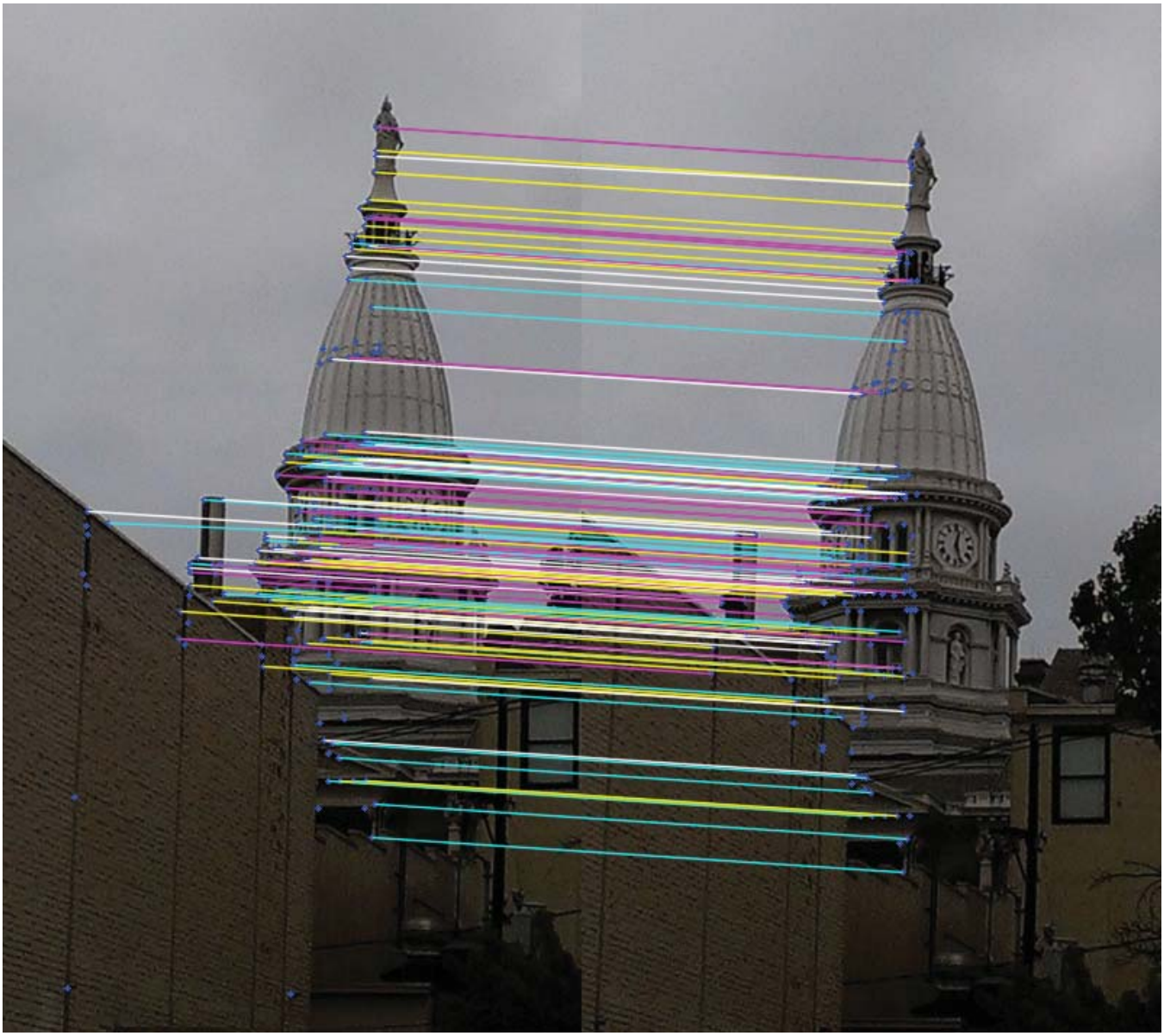

Figure 8: NCC Feature Matching on tower1.jpg, tower2.jpg - 128 correspondences found

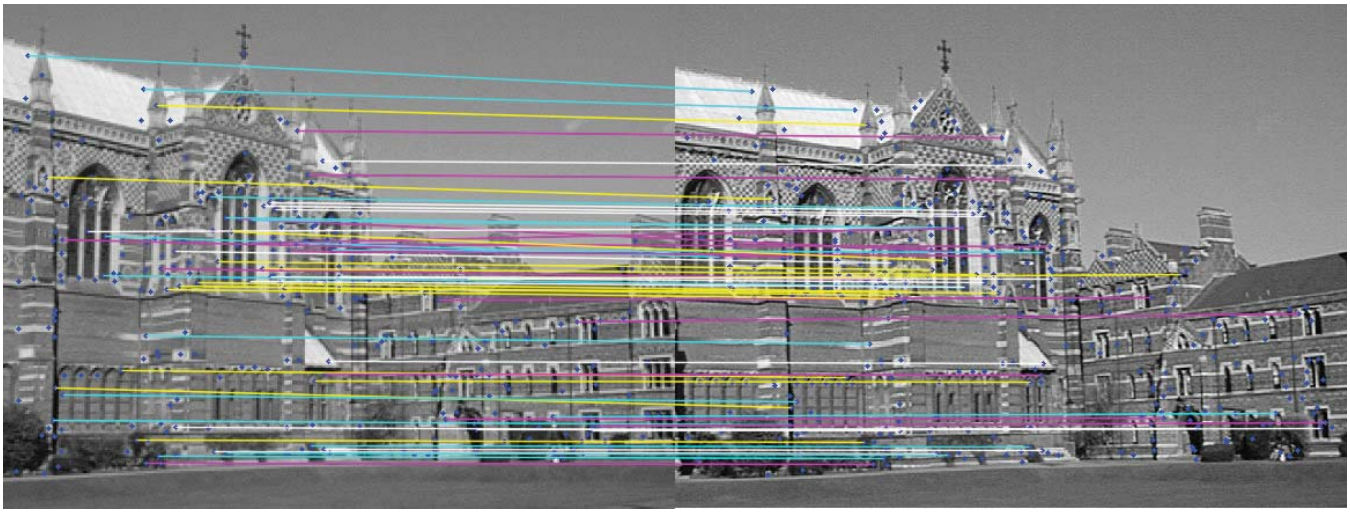

Figure 9: SURF on building1.jpg, building2.jpg - 58 correspondences found

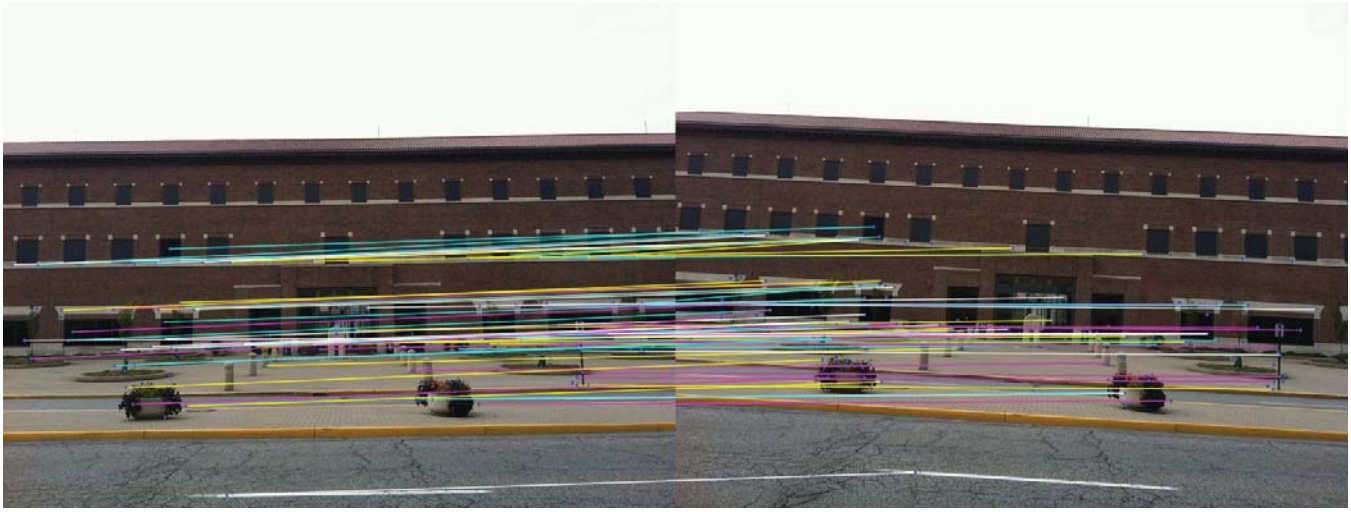

Figure 10: SURF on sample-a.jpg, sample-b.jpg - 64 correspondences found

Figure 11: SURF on stereo.jpg, stereo1.jpg - 10 correspondences found but lots of false positives

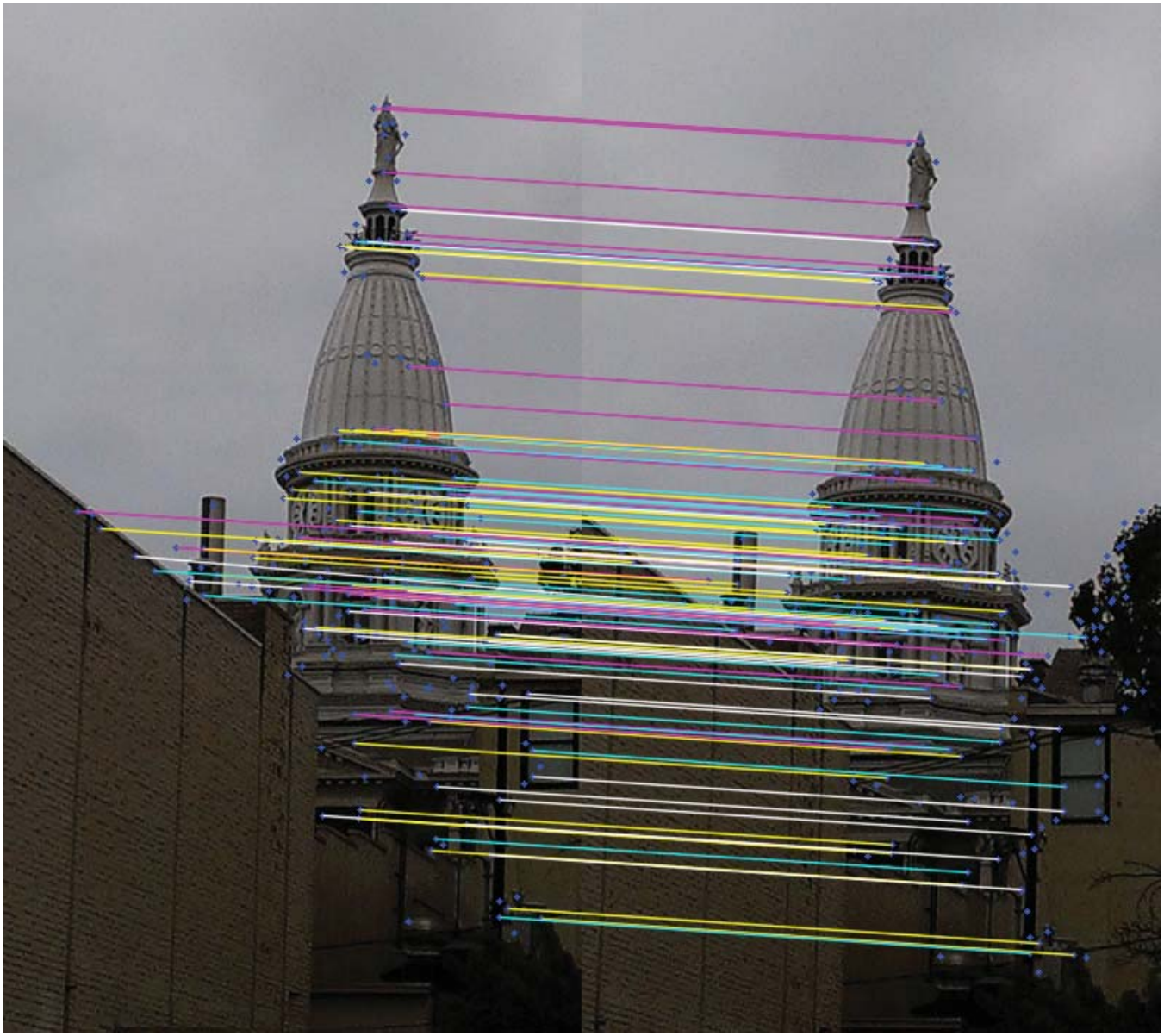

Figure 12: SURF on tower1.jpg, tower2.jpg - 115 correspondences found

7. Source Code: harris_corner.py

```
#!/usr/bin/python
#
# Author: Sriram Karthik Badam
# Date: Sep 26, 2012
#

import sys, os
import cv
import numpy as np

#image variables
image1 = 0
image2 = 0

#constants
WINDOW_SIZE = 5
TOTAL_FEATURES = 200


#Window size and Thresholds for stereo.jpg
"""
SSD_WINDOW_SIZE = 41
NCC_WINDOW_SIZE = 53
SSD_THRESHOLD = 1100
SSD_RATIO_THRESHOLD = 0.8
NCC_THRESHOLD = 0.8
NCC_RATIO_THRESHOLD = 1.2
"""

#Window size and Thresholds for building.jpg
"""
SSD_WINDOW_SIZE = 41
NCC_WINDOW_SIZE = 15
SSD_THRESHOLD = 1300
SSD_RATIO_THRESHOLD = 0.85
NCC_THRESHOLD = 0.75
NCC_RATIO_THRESHOLD = 1.2
"""

#Window size and Thresholds for sample .jpg

SSD_WINDOW_SIZE = 41
NCC_WINDOW_SIZE = 53
SSD_THRESHOLD = 1100
SSD_RATIO_THRESHOLD = 0.75
NCC_THRESHOLD = 0.78
NCC_RATIO_THRESHOLD = 1.1

#Window size and Thresholds for tower.jpg
"""
SSD_WINDOW_SIZE = 41
NCC_WINDOW_SIZE = 53
SSD_THRESHOLD = 1300
SSD_RATIO_THRESHOLD = 0.7
NCC_THRESHOLD = 0.76
NCC_RATIO_THRESHOLD = 1.07
```

```
"""

#input algorithm = 0 => SSD ; algorithm = 1 => NCC
algorithm = 0
size = 0#input algorithm window size

#feature class
class Feature(object):
  #initializing
  def __init__(self, x, y):
    self.x = x
    self.y = y
    self.match = -1
    self.score = -1
    self.neighbor = 0


"""
Corner Detector function calculates the Corner Responses for
each pixel of the image. By applying non-maximum suppression
we get the interest points
"""

def harrisCornerDetector(image):

  #image derivative along x
  image_der_x = cv.CreateImage((image.width, image.height), cv.IPL_DEPTH_32F, 1)
  #image derivative along y
  image_der_y = cv.CreateImage((image.width, image.height), cv.IPL_DEPTH_32F, 1)

  #order of derivative in x
  order_x = 1
  #order of derivative in y
  order_y = 1
  #Operator Size
  aperture_size = 3 #using 3x3 sobel operator

  #grey scale
  gray_scale_image1 = cv.CreateImage((image.width, image.height), image.depth, 1)
  cv.CvtColor(image, gray_scale_image1, cv.CV_RGB2GRAY)

  #Applies Sobel Operator to get the image derivatives along x, y
  cv.Sobel(gray_scale_image1, image_der_x, order_x, 0, aperture_size)
  cv.Sobel(gray_scale_image1, image_der_y, 0, order_y, aperture_size)

  #Calculates C
  global WINDOW_SIZE #size of mask
  C = cv.CreateMat(2, 2, cv.CV_64FC1)
  #variables for SVD
  U = cv.CreateMat(2, 2, cv.CV_64FC1)
  D = cv.CreateMat(2, 2, cv.CV_64FC1)

  I_x = 0
  I_xy = 0
  I_y = 0
  R = cv.CreateMat(image.width, image.height, cv.CV_64FC1)
  cv.Zero(R)
```

```
  #Constructs the variance Matrix at each pixel
  #loops through each pixel
  print "Calculating the Covariance"
  for i in range(int(WINDOW_SIZE/2), image.width - int(WINDOW_SIZE/2 + 1)):
    for j in range(int(WINDOW_SIZE/2), image.height - int(WINDOW_SIZE/2 + 1)):
      I_x = 0
      I_xy = 0
      I_y = 0
      for index_x in range(-int(WINDOW_SIZE/2), int(WINDOW_SIZE/2 + 1)):
  for index_y in range(-int(WINDOW_SIZE/2), int(WINDOW_SIZE/2 + 1)):
    I_x = I_x + image_der_x[j + index_y, i + index_x] * image_der_x[j + index_y, i +
        index_x] / WINDOW_SIZE
    I_xy = I_xy + image_der_x[j + index_y, i + index_x] * image_der_y[j + index_y, i +
         index_x] / WINDOW_SIZE
    I_y = I_y + image_der_y[j + index_y, i + index_x] * image_der_y[j + index_y, i +
        index_x] / WINDOW_SIZE

      cv.mSet(C, 0, 0, I_x)
      cv.mSet(C, 1, 0, I_xy)
      cv.mSet(C, 0, 1, I_xy)
      cv.mSet(C, 1, 1, I_y)

      #svd
      cv.SVD(C, D, U, None, 0)
      lambda1 = cv.mGet(D, 0, 0)
      lambda2 = cv.mGet(D, 1, 1)
      #corner Response
      cv.mSet(R, i, j, lambda1*lambda2 - 0.04*np.power((lambda1+lambda2), 2))

  R_final = cv.CreateMat(image.width, image.height, cv.CV_64FC1)
  cv.Copy(R, R_final)

  #applys non maximum suppression -> elements other than maximum values in a window
      are removed
  for i in range(int(WINDOW_SIZE/2), image.width-int(WINDOW_SIZE/2)):
    for j in range(int(WINDOW_SIZE/2), image.height-int(WINDOW_SIZE/2)):
      for index_x in range(-int(WINDOW_SIZE/2), int(WINDOW_SIZE/2 + 1)):
  for index_y in range(-int(WINDOW_SIZE/2), int(WINDOW_SIZE/2 + 1)):
    R_value = cv.mGet(R, i, j)
    R_neighbor_value = cv.mGet(R, i + index_x, j + index_y)
    if R_value!= 0 and R_value < R_neighbor_value:
      cv.mSet(R_final, i, j, 0)
      break
  return R_final


"""
Creates the feature descriptor for each interest point
"""
def getFeatures(R, image, features, threshold):
  global algorithm
  global TOTAL_FEATURES
  global size

  if algorithm == 0:
    size = SSD_WINDOW_SIZE
```

```
  if algorithm == 1:
    size = NCC_WINDOW_SIZE

  feature_number = 0
  for i in range(int(size/2), image.width-int(size/2)):
    for j in range(int(size/2), image.height-int(size/2)):
      if cv.mGet(R, i, j) > threshold: #threshold on the corner response to prune
          interest points
  if feature_number < TOTAL_FEATURES:
    feature_number = feature_number + 1
    features.append(Feature(i, j)) #adds the feature to the feature list
  print "number of features:", feature_number


"""
fills the 'neighbor' variable with the neighboring pixel values of a given pixel.
"""
def getNeighbor(x, y, grey_scale_image, neighbor):
  global size
  for i in range(-int(size/2), int(size/2+1)):
    for j in range(-int(size/2), int(size/2+1)):
      cv.mSet(neighbor, int(size/2) + j, int(size/2) + i, grey_scale_image[y+j, x+i])


"""
Computes the SSD Score
"""
def ssdScore(f1, f2):
  global size #size of SSD window
  #subtracts f2 from f1
  sub_f1_f2 = cv.CreateMat(size, size, cv.CV_64FC1)
  cv.Sub(f1, f2, sub_f1_f2)

  #square and add
  f1_f2_square = cv.CreateMat(size, size, cv.CV_64FC1)
  cv.Pow(sub_f1_f2, f1_f2_square, 2)
  score = cv.Sum(f1_f2_square)
  return score[0]/(size*size)

"""
Computes the NCC Score
"""
def nccScore(f1, f2):
  global size #size of NCC window
  mean1 = cv.Avg(f1)
  mean2 = cv.Avg(f2)
  f1_sub_mean = cv.CreateMat(size, size, cv.CV_64FC1)
  f2_sub_mean = cv.CreateMat(size, size, cv.CV_64FC1)
  cv.SubS(f1, mean1, f1_sub_mean);
  cv.SubS(f2, mean2, f2_sub_mean);

  #calculates numerator of the ncc score fraction
  ncc_numerator = cv.CreateMat(size, size, cv.CV_64FC1)
  cv.Mul(f1_sub_mean, f2_sub_mean, ncc_numerator)
  numerator = cv.Sum(ncc_numerator)

  #calculates denominator
```

```
  magnitude1 = cv.CreateMat(size, size, cv.CV_64FC1)
  magnitude2 = cv.CreateMat(size, size, cv.CV_64FC1)
  cv.Pow(f1_sub_mean, magnitude1, 2)
  sum1 = cv.Sum(magnitude1)
  cv.Pow(f2_sub_mean, magnitude2, 2)
  sum2 = cv.Sum(magnitude2)
  denominator = np.sqrt(sum1[0]*sum2[0])

  score = numerator[0]/denominator
  return score

"""
prints a matrix
"""
def printMatrix(mat):
  temp_array = np.asarray(mat[:,:])
  print temp_array


"""
main function
"""
def main():
  print "@Author: S.Karthik Badam"
  global image1
  global image2

  #loads image
  if len(sys.argv) > 2:
    filename1 = sys.argv[1]
    filename2 = sys.argv[2]
    image1 = cv.LoadImage(filename1, cv.CV_LOAD_IMAGE_UNCHANGED)
    image2 = cv.LoadImage(filename2, cv.CV_LOAD_IMAGE_UNCHANGED)
    if image1 == 0:
      print "enter a valid Image Path for Image 1"
    if image2 == 0:
      print "enter a valid Image Path for Image 2"
  else:
    print "Enter two valid image file paths as argument"

  #applies corner Detector
  R1 = harrisCornerDetector(image1)
  R2 = harrisCornerDetector(image2)

  #asks user for the algorithm choice
  print "type 0 to select SSD and 1 to select NCC"
  global algorithm
  algorithm = int(raw_input())

  #Creates feature Descriptors
  features1 = []
  features2 = []

  #fills the feature array
  #building .jpg
  """
  getFeatures(R1, image1, features1, 7e9)
```

```
getFeatures(R2, image2, features2, 3e10)
"""
#sample .jpg

getFeatures(R1, image1, features1, 7e9)
getFeatures(R2, image2, features2, 10e9)

#others
"""
getFeatures(R1, image1, features1, 7e8)
getFeatures(R2, image2, features2, 8e8)
"""

grey_scale_image1 = cv.CreateImage((image1.width, image1.height), image1.depth, 1)
grey_scale_image2 = cv.CreateImage((image2.width, image2.height), image2.depth, 1)
cv.CvtColor(image1, grey_scale_image1, cv.CV_RGB2GRAY)
cv.CvtColor(image2, grey_scale_image2, cv.CV_RGB2GRAY)

global size, SSD_THRESHOLD, SSD_RATIO_THRESHOLD
global NCC_THRESHOLD, NCC_RATIO_THRESHOLD

#Computes neighbors of each interest point and puts them in the feature descriptor
global TOTAL_FEATURES
for i in range(TOTAL_FEATURES):
  f1 = cv.CreateMat(size, size, cv.CV_64FC1)
  f2 = cv.CreateMat(size, size, cv.CV_64FC1)

  if i < len(features1):
    x1 = features1[i].x
    y1 = features1[i].y
    getNeighbor(x1, y1, grey_scale_image1, f1)
    features1[i].neighbor = f1

  if i < len(features2):
    x2 = features2[i].x
    y2 = features2[i].y
    getNeighbor(x2, y2, grey_scale_image2, f2)
    features2[i].neighbor = f2

#Finds Correspondences

#If SSD Chosen
if algorithm == 0:
  for i in range(len(features1)):
    score = 0
    score_min = 1e10
    score_second_min = 1e10
    for j in range(len(features2)):
score = ssdScore(features1[i].neighbor, features2[j].neighbor)
#updates min, second_min
if score_min > score and score < SSD_THRESHOLD:#SSD_THRESHOLD is a threshold on ssd
    score values
  score_second_min = score_min
  score_min = score
  features1[i].score = score_min
  features1[i].match = j
  features2[j].score = score_min
```

```
  features2[j].match = i
#updates second_min
elif score < score_second_min and score > score_min:
  score_second_min = score

    if score_second_min > 0 and score_min/score_second_min > SSD_RATIO_THRESHOLD: #
       Threshold on the ratio
features1[i].score = -1
features1[i].match = -1
features2[j].score = -1
features2[j].match = -1

#if NCC chosen
if algorithm == 1:
  for i in range(len(features1)):
    score = 0
    score_max = -1e10
    score_second_max = -1e10
    for j in range(len(features2)):
      score = nccScore(features1[i].neighbor, features2[j].neighbor)
#updates max, second_max
if score > score_max and score > NCC_THRESHOLD:#NCC_THESHOLD is a threshold on the
   NCC score Values
  score_second_max = score_max
  score_max = score
  features1[i].score = score_max
  features1[i].match = j
  features2[j].score = score_max
  features2[j].match = i
#updates second_max
elif score > score_second_max and score < score_max:
  score_second_max = score

    if score_second_max > 0 and score_max/score_second_max < NCC_RATIO_THRESHOLD:#
       threshold on ratio
features1[i].score = -1
features1[i].match = -1
features2[j].score = -1
features2[j].match = -1


#reconstruct the final image
final_image = cv.CreateImage((2*image1.width, image1.height), image1.depth, 3)

cv.SetImageROI(final_image, (0, 0, image1.width, image1.height))
cv.Copy(image1, final_image)

#marks the interest points in the first part
for i in range(len(features1)):
  cv.Circle(final_image, (features1[i].x, features1[i].y), 0, (255, 0, 0), 4)
cv.ResetImageROI(final_image)

cv.SetImageROI(final_image, (image1.width, 0, image1.width, image1.height))
cv.Copy(image2, final_image)

#marks the interest points in the second part of the image
for i in range(len(features2)):
```

```
    cv.Circle(final_image, (features2[i].x, features2[i].y), 0, (255, 0, 0), 4)
  cv.ResetImageROI(final_image)

  #shows correspondences by drawing a line between matching Interest Points
  count = 0
  for i in range(len(features1)):
    if features1[i].match != -1:
      count = count + 1
      #different color for each line to differentiate between them.
      cv.Line(final_image, (features1[i].x, features1[i].y), (features2[features1[i].
          match].x + image1.width, features2[features1[i].match].y), (255*(count%4),
          255*((count+1)%4), 255*((count+2)%4)), 1, cv.CV_AA, 0)

  print "Number of Correspondences = ", count
  cv.SaveImage("Result.png", final_image)#result stored in Result.png

if __name__=="__main__":
  main()
```

8. Source Code: SURF.py

```
#!/usr/bin/python
#
# Author: Sriram Karthik Badam
# Date: Sep 26, 2012
#

import sys, os
import cv
import numpy as np

#image variables
image1 = 0
image2 = 0

#Thresholds for stereo .jpg
"""
SCORE_THRESHOLD = 0.2
RATIO_THRESHOLD = 0.9
SURF_THRESHOLD = 1000
"""

#Thresholds for tower .jpg
"""
SCORE_THRESHOLD = 0.15
RATIO_THRESHOLD = 0.8
SURF_THRESHOLD = 450
"""

#Thresholds for building .jpg
"""
SCORE_THRESHOLD = 0.15
RATIO_THRESHOLD = 0.9
SURF_THRESHOLD = 2000
"""

#Thresholds for sample .jpg
SCORE_THRESHOLD = 0.15
RATIO_THRESHOLD = 0.9
SURF_THRESHOLD = 2000
"""
Computes the feature points and feature descriptors using the Opencv ExtractSURF
    method.
"""

def SURFdetector(image):
  global SURF_THRESHOLD #Threshold on the Hessian Value to prune the interest points
  (feature_points, feature_descriptors) = cv.ExtractSURF(image, None, cv.
      CreateMemStorage(), (0, SURF_THRESHOLD, 3, 1))
  return (feature_points, feature_descriptors)

"""
Computes the distance between two feature descriptors using Euclidean Distance measure
"""
def distance(p1, p2):
  p1_sub_p2 = np.subtract(p1, p2)
  p1_p2_square = np.power(p1_sub_p2, 2)
```

```
  score = np.sum(p1_p2_square)
  return np.power(score, 0.5)


"""
main function
"""
def main():
  print "@Author: S.Karthik Badam"
  global image1
  global image2

  #loads image
  if len(sys.argv) > 2:
    filename1 = sys.argv[1]
    filename2 = sys.argv[2]
    image1 = cv.LoadImage(filename1, cv.CV_LOAD_IMAGE_UNCHANGED)
    image2 = cv.LoadImage(filename2, cv.CV_LOAD_IMAGE_UNCHANGED)
    if image1 == 0:
      print "enter a valid Image Path for Image 1"
    if image2 == 0:
      print "enter a valid Image Path for Image 2"
  else:
    print "Enter two valid image file paths as argument"

  grey_scale_image1 = cv.CreateImage((image1.width, image1.height), image1.depth, 1)
  grey_scale_image2 = cv.CreateImage((image2.width, image2.height), image2.depth, 1)
  cv.CvtColor(image1, grey_scale_image1, cv.CV_RGB2GRAY)
  cv.CvtColor(image2, grey_scale_image2, cv.CV_RGB2GRAY)

  #gets the feature points and feature descriptors
  (feature_points1, feature_descriptors1) = SURFdetector(grey_scale_image1)
  (feature_points2, feature_descriptors2) = SURFdetector(grey_scale_image2)
  correspondence = cv.CreateMat(len(feature_points1), 1, cv.CV_64FC1)

  print "number of features in image 1:", len(feature_points1)
  print "number of features in image 2:", len(feature_points2)

  global SCORE_THRESHOLD, RATIO_THRESHOLD

  #Calculates the correspondences
  for i in range(len(feature_points1)):
    score = 0
    score_min = 1e10
    score_second_min = 1e10
    for j in range(len(feature_points2)):
      score =  distance(feature_descriptors1[i], feature_descriptors2[j])
      #updates the min, second_min
      if score_min > score and score < SCORE_THRESHOLD:
  score_second_min = score_min
  score_min = score
  cv.mSet(correspondence, i, 0, j)
      #updates second_min
      elif score_min < score and score > score_second_min:
  score_second_min = score

    if score_second_min > 0 and score_min/score_second_min > RATIO_THRESHOLD:
```

```
    cv.mSet(correspondence, i, 0, -1)

    #Creates the result image
    final_image = cv.CreateImage((2*image1.width, image1.height), image1.depth, 3)
    cv.SetImageROI(final_image, (0, 0, image1.width, image1.height))
    cv.Copy(image1, final_image)

    #marks the Interest Points in the first part of the image
    for ((x,y), laplacian, size, dir, hessian) in feature_points1:
      cv.Circle(final_image, (int(x), int(y)), 0, (255, 0, 0), 4)

    cv.ResetImageROI(final_image)
    cv.SetImageROI(final_image, (image1.width, 0, image1.width, image1.height))
    cv.Copy(image2, final_image)

    #marks the Interest Points in the second part of the image
    for ((x,y), laplacian, size, dir, hessian) in feature_points2:
      cv.Circle(final_image, (int(x), int(y)), 0, (255, 0, 0), 4)

    cv.ResetImageROI(final_image)

    #Draws lines between the correspondences to show them on the final image
    count = 0
    for i in range(len(feature_points1)):
      value = cv.mGet(correspondence, i,0)
      if value != -1:
        ((x1,y1), laplacian1, size1, dir, hessian1) = feature_points1[i]
        ((x2,y2), laplacian2, size2, dir, hessian2) = feature_points2[int(value)]
        count = count + 1
        #Multiple line colors are used to avoid confusion with overlaps
        cv.Line(final_image, (int(x1),int(y1)), (int(x2)+image1.width,int(y2)), (255*(
            count%4), 255*((count+1)%4), 255*((count+2)%4)), 1, cv.CV_AA, 0)

    print "Number of Correspondences: ", count
    cv.SaveImage("Result_SURF.png", final_image)

if __name__=="__main__":
    main()
```