CS 682: COMPUTER VISION

Abhishek Bodas

G01160204

HW4

Website: http://mason.gmu.edu/~abodas/vision/

Username: CS682

Password: abodas682

1. The features of GaitImages are calculated using the respective OpenCV functions. The image is converted to binary at the beginning and then it is used to compute contours, contour area and perimeter, convex hull, polygonal approximations, deficits of convexity, moments.

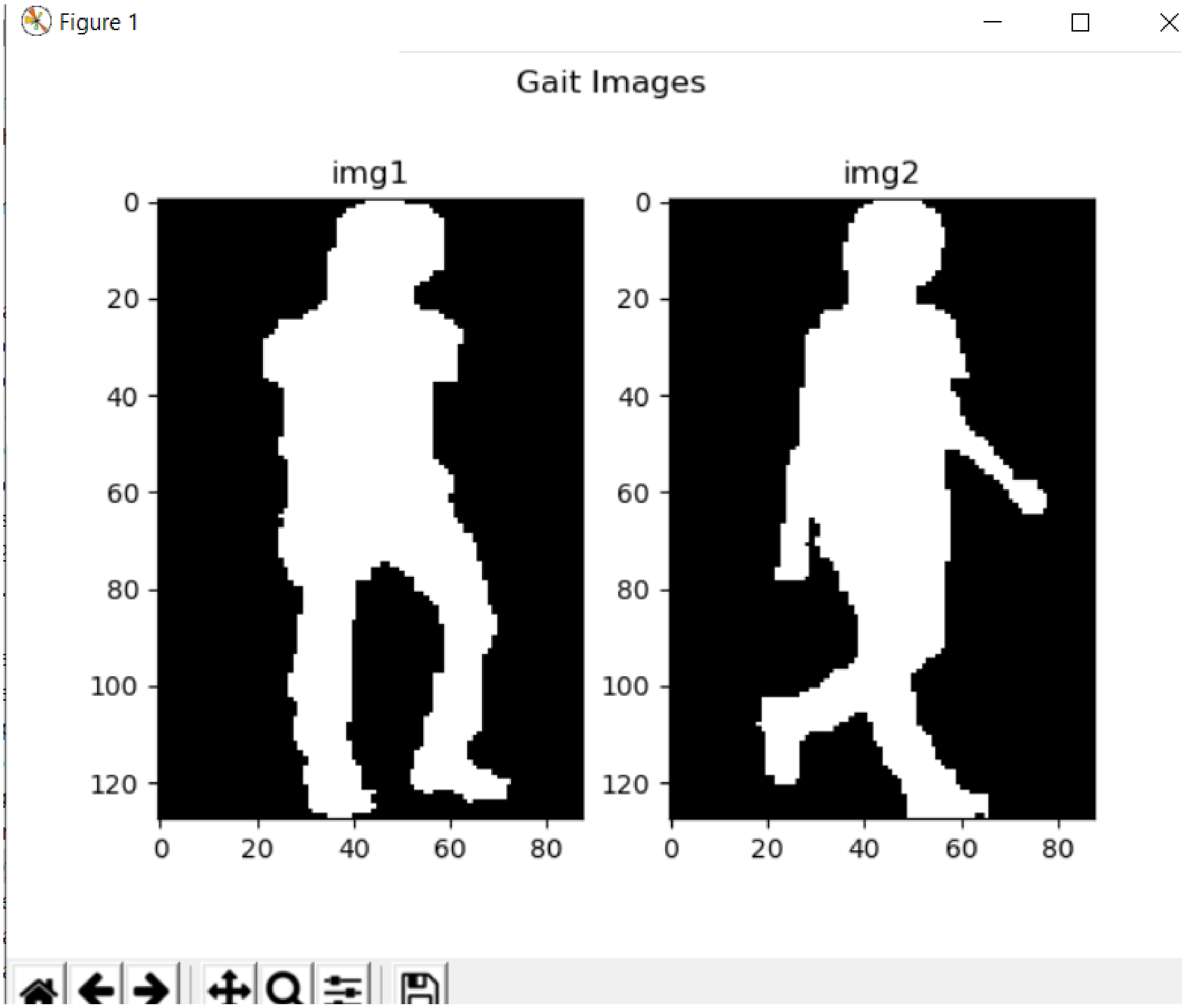
   Code:

```
#Script to display various shape descriptors and compute curvature,distance transform, chamfer
matching of gait images
import cv2
import glob #To import glob module
import numpy as np
from matplotlib import pyplot as plt
import math
from prettytable import PrettyTable

imglist = []
print("Binary silhouettes")
for eachimage in glob.glob('GaitImages/*.png'): #Location of unzipped image folder
  print(eachimage)
  image = cv2.imread(eachimage)
  imglist.append(image) #To create a list of all gait images
#To select 2 images
imgselect = [cv2.threshold(cv2.cvtColor(imglist[0], cv2.COLOR_BGR2GRAY), 127, 255,
cv2.THRESH_BINARY)[1],
cv2.threshold(cv2.cvtColor(imglist[25],cv2.COLOR_BGR2GRAY), 127, 255,
cv2.THRESH_BINARY)[1]]
indexes=[0,25]

def Imageplot(img1,img2,description): #Function to plot images
  figure, ax = plt.subplots(1, 2)
```

```
    ax[0].imshow(img1, cmap="gray")
    ax[0].set_title("img1")
    ax[1].imshow(img2, cmap="gray")
    ax[1].set_title("img2")
    plt.suptitle(description)
    plt.show()

Imageplot(imgselect[0],imgselect[1],"Gait Images")

contours = []
contourlist = []
contourarea = []
contourperimeter = []
polygonapprox = []
imgpolygon = []
convexhull = []
convexhullindex = []
imgconvexhull =[]
defects = []
defectpoints = []
imgdefects = [imglist[0].copy(), imglist[25].copy()]
convexhullarea = []
convexhullperimeter = []
defectsarea = []
defectsperimeter = []
imgmoments = []
hullmoments = []

for i,index in enumerate(indexes): #To find and apply contours
    contour,hierarchy = cv2.findContours(imgselect[i],cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)
    contours.append(contour[0])
    contourlist.append(cv2.drawContours(imglist[index].copy(), [contours[i]], -1, (0, 255, 255)))

contourarea.append(cv2.contourArea(contours[0]))
contourarea.append(cv2.contourArea(contours[1])) #Contour Area
contourperimeter.append(cv2.arcLength(contours[0],True))
contourperimeter.append(cv2.arcLength(contours[1],True)) #Contour Perimeter

for i,index in enumerate(indexes):
    polygonapprox.append(cv2.approxPolyDP(contours[i], 0.01*contourperimeter[i], True))
#Polygonal Approximation
    imgpolygon.append(cv2.drawContours(imglist[index].copy(), [polygonapprox[i]], -1, (0, 255,
255),1))
```

```
    convexhull.append(cv2.convexHull(contours[i]))
    convexhullindex.append(cv2.convexHull(contours[i], returnPoints = False))
    imgconvexhull.append(cv2.drawContours(imglist[index].copy(), [convexhull[i]], -1, (0, 255,
255),1))#ConvexHull
    defects.append(cv2.convexityDefects(contours[i], convexhullindex[i]))
    deficitpoints = []
    for j in range(defects[i].shape[0]): # Convexity defects
        s,e,f,d = defects[i][j,0]
        start = tuple(contours[i][s][0])
        end = tuple(contours[i][e][0])
        far = tuple(contours[i][f][0])
        deficitpoints.append(list(contours[i][f][0]))
        cv2.line(imgdefects[i],start,end,[0,255,255],1)
        cv2.circle(imgdefects[i],far,2,[0,255,0],-1)
    defectpoints.append(np.array([deficitpoints]))

convexhullarea.append(cv2.contourArea(convexhull[0]))
convexhullarea.append(cv2.contourArea(convexhull[1]))
convexhullperimeter.append(cv2.arcLength(convexhull[0], True))
convexhullperimeter.append(cv2.arcLength(convexhull[1], True))
defectsarea.append(cv2.contourArea(defectpoints[0])) #Convexity defect area
defectsarea.append(cv2.contourArea(defectpoints[1]))
defectsperimeter.append(cv2.arcLength(defectpoints[0], True))
defectsperimeter.append(cv2.arcLength(defectpoints[1], True)) # Convexity defect Perimeter
imgmoments.append(sorted(cv2.moments(imgselect[0]).items()))
imgmoments.append(sorted(cv2.moments(imgselect[1]).items()))  #Image Moments
hullmoments.append(sorted(cv2.moments(convexhull[0]).items()))
hullmoments.append(sorted(cv2.moments(convexhull[1]).items())) #Hull Moments


print("Area1:",
contourarea[0],"Area2:",contourarea[1],"\n","Perimeter1:",contourperimeter[0],"Perimeter2:",c
ontourperimeter[1],"\n", "Hull Area1:",
      convexhullarea[0],"Hull Area2:", convexhullarea[1],"\n","Hull Perimeter1:",
convexhullperimeter[0], "Hull Perimeter2:", convexhullperimeter[1],"\n",
      "Deficit1:", len(defects[0]),"Deficit Area1:",defectsarea[0],"Deficit
Perimeter1:",defectsperimeter[0],"\n", "Deficit2:",len(defects[1]),
      "Deficit Area2:",defectsarea[1],"Deficit Perimeter2:",defectsperimeter[1],"\n","Image
Moments1:",str(imgmoments[0][:9])+"\n",
      "Image Moments2:",str(imgmoments[1][:9])+"\n", "Hull
Moments1:",str(hullmoments[0][:9])+"\n","Hull Moments2:",str(hullmoments[1][:9])+"\n")
```

Output:



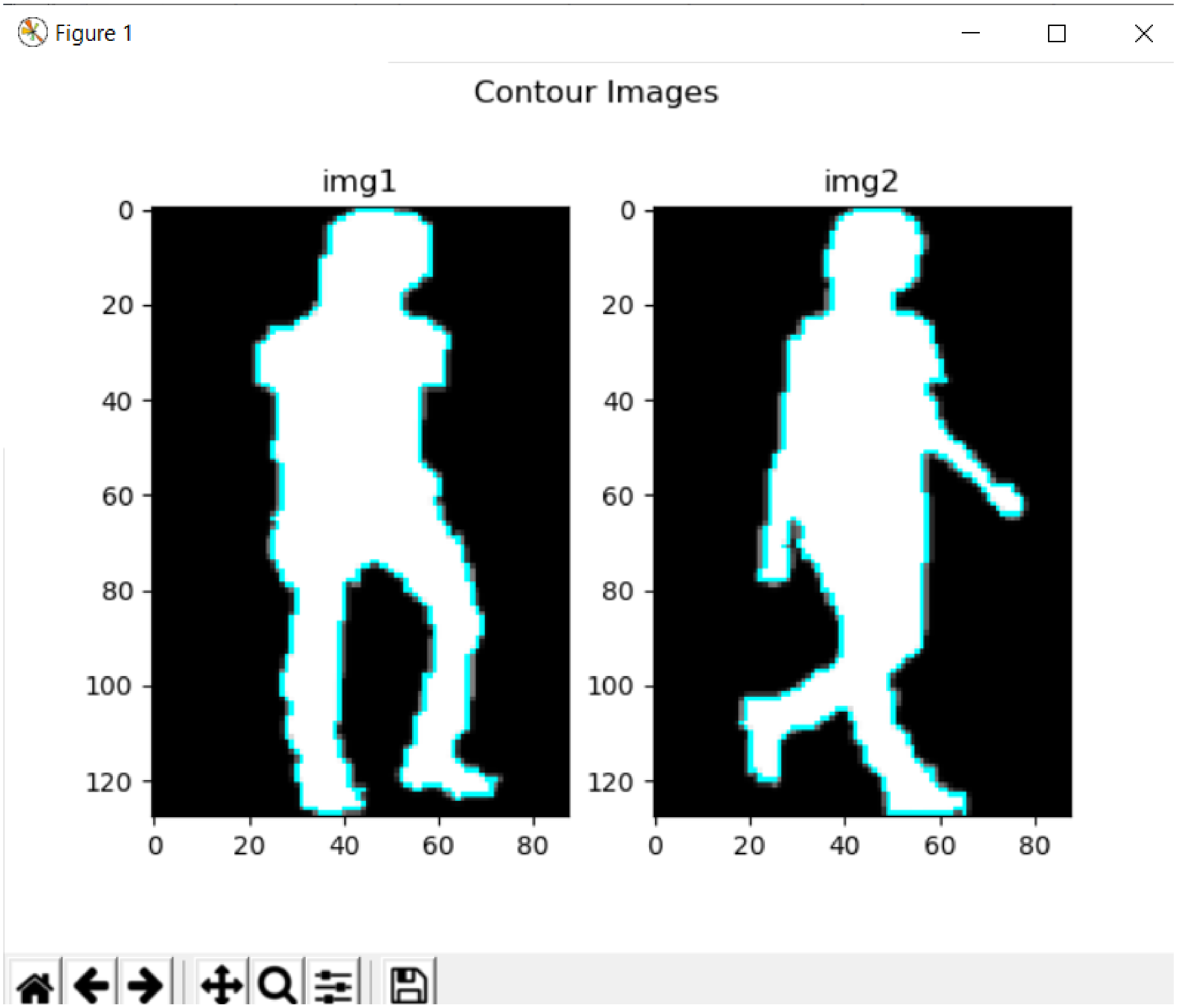


Figure 1
Contour Images
img1
img2
0
20
40
60
80
100
120
0
20
40
60
80

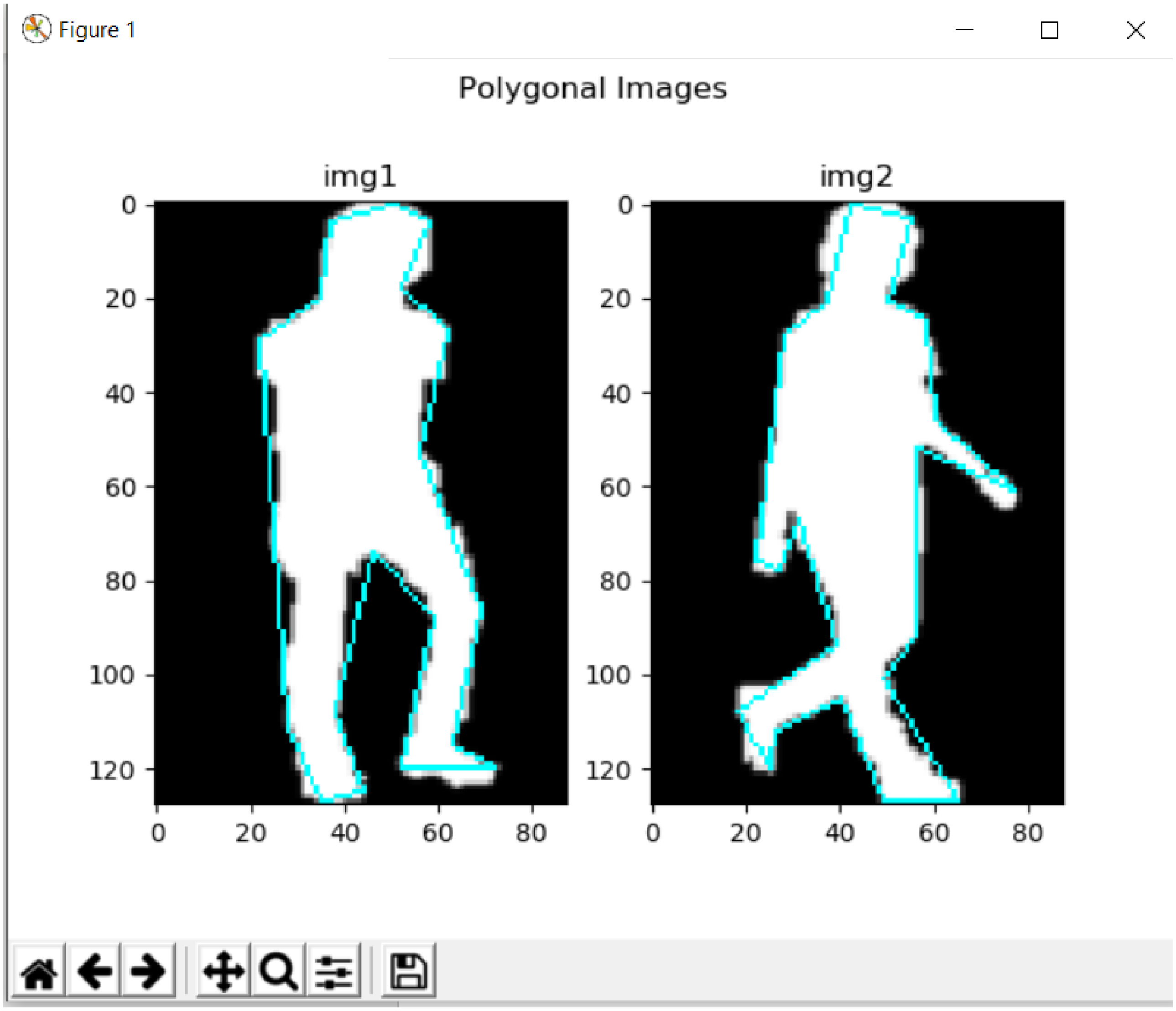


Figure 1
Polygonal Images
img1
img2
0
20
40
60
80
100
120
0
20
40
60
80

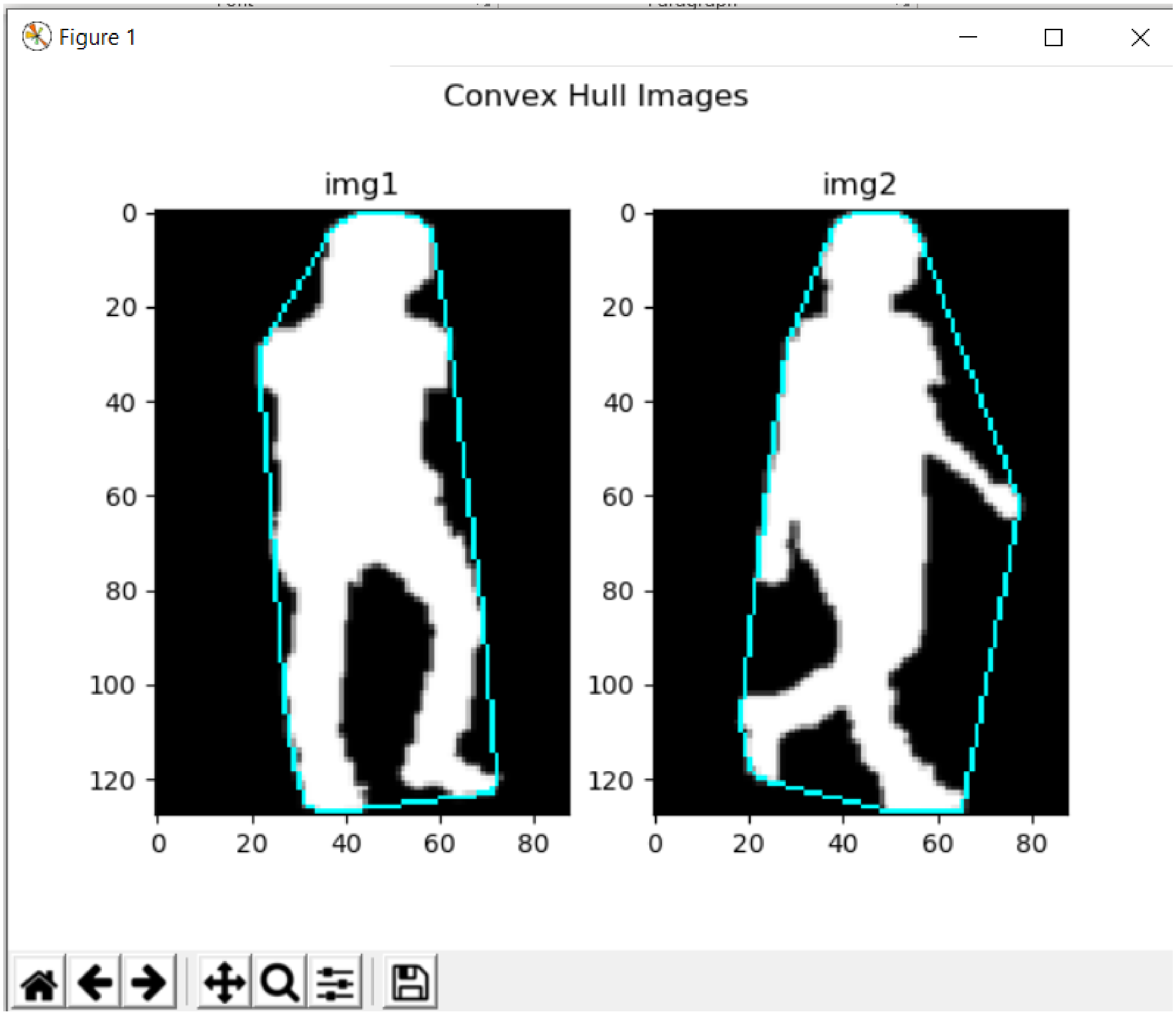


Figure 1
Convex Hull Images
img1
img2
0
20
40
60
80
100
120
0
20
40
60
80

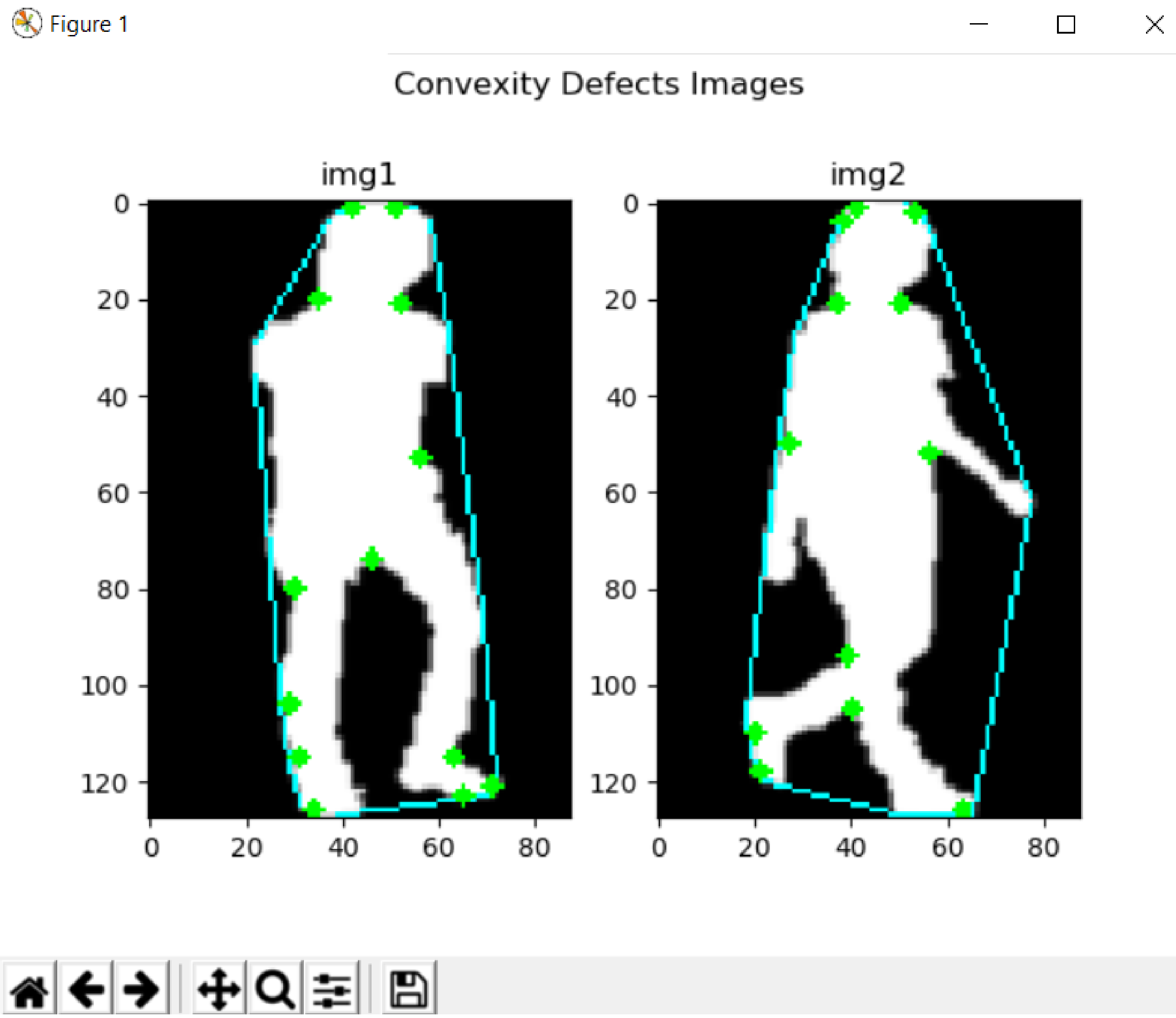


Figure 1
Convexity Defects Images
img1
img2
0
20
40
60
80
100
120
0
20
40
60
80

2. All the feature values are displayed in the table using prettytable module

   Code:

```
#Creating a table to display values
attribute = ['ContourArea1','ContourArea2','ContourPerimeter1','ContourPerimeter2','HullArea1'
,'HullArea2', 'HullPerimeter1',

'HullPerimeter2','Deficits1','DeficitArea1','DeficitPerimeter1','Deficits2','DeficitArea2','DeficitPerim
eter2']
value =
[contourarea[0],contourarea[1],contourperimeter[0],contourperimeter[1],convexhullarea[0],con
vexhullarea[1],convexhullperimeter[0],

convexhullperimeter[1],len(defects[0]),defectsarea[0],defectsperimeter[0],len(defects[1]),defect
sarea[1],defectsperimeter[1]]
table = PrettyTable(['attribute','value'])
for i in range (0,14):
   table.add_row([attribute[i],value[i]])
print(table)

#Table for moments
moments = ['ImageMoments1','ImageMoments2','HullMoments1','HullMoments2']
momentvalue =
[str(imgmoments[0][:9]),str(imgmoments[1][:9]),str(hullmoments[0][:9]),str(hullmoments[1][:9])
]
tab = PrettyTable(['moments','momentvalue'])
for i in range (0,4):
   tab.add_row([moments[i],momentvalue[i]])
         print(tab)
```

Output:

attribute	value
ContourArea1	3299.0
ContourArea2	3053.0
ContourPerimeter1	470.8771975040436
ContourPerimeter2	475.0193328857422
HullArea1	4908.0
HullArea2	5448.5
HullPerimeter1	310.23964416980743
HullPerimeter2	312.7183817625046
Deficits1	13
DeficitArea1	2268.5
DeficitPerimeter1	372.973219871521
Deficits2	12
DeficitArea2	2614.5
DeficitPerimeter2	321.8056926727295

moments	momentvalue
ImageMoments1	[('m00', 895305.0), ('m01', 54892065.0), ('m02', 4382169645.0), ('m03', 398494075575.0), ('m10', 39937080.0), ('m11', 2488431015.0), ('m12', 201839071095.0), ('m20', 1901116800.0), ('m21', 121523795775.0)]
ImageMoments2	[('m00', 832575.0), ('m01', 51059160.0), ('m02', 4024215690.0), ('m03', 361221217470.0), ('m10', 36221985.0), ('m11', 2199546105.0), ('m12', 172932434805.0), ('m20', 1668931905.0), ('m21', 101249089005.0)]
HullMoments1	[('m00', 4908.0), ('m01', 329913.1666666666), ('m02', 27905875.333333332), ('m03', 2629436872.15), ('m10', 226410.66666666666), ('m11', 15547724.5), ('m12', 1335363179.3833332), ('m20', 11147330.0), ('m21', 783093425.35)]
HullMoments2	[('m00', 5448.5), ('m01', 378145.0), ('m02', 32036905.583333332), ('m03', 2995639701.3), ('m10', 255586.5), ('m11', 17598906.125), ('m12', 1478479520.5333333), ('m20', 13016860.083333332), ('m21', 896901379.6)]

3. Curvature

Curvature is calculated using the contour points. It is interpolated using the second order polynomial formula:

x(t) = a0 + a1 t + a2t2
y(t) = b0 + b1 t + b2t2

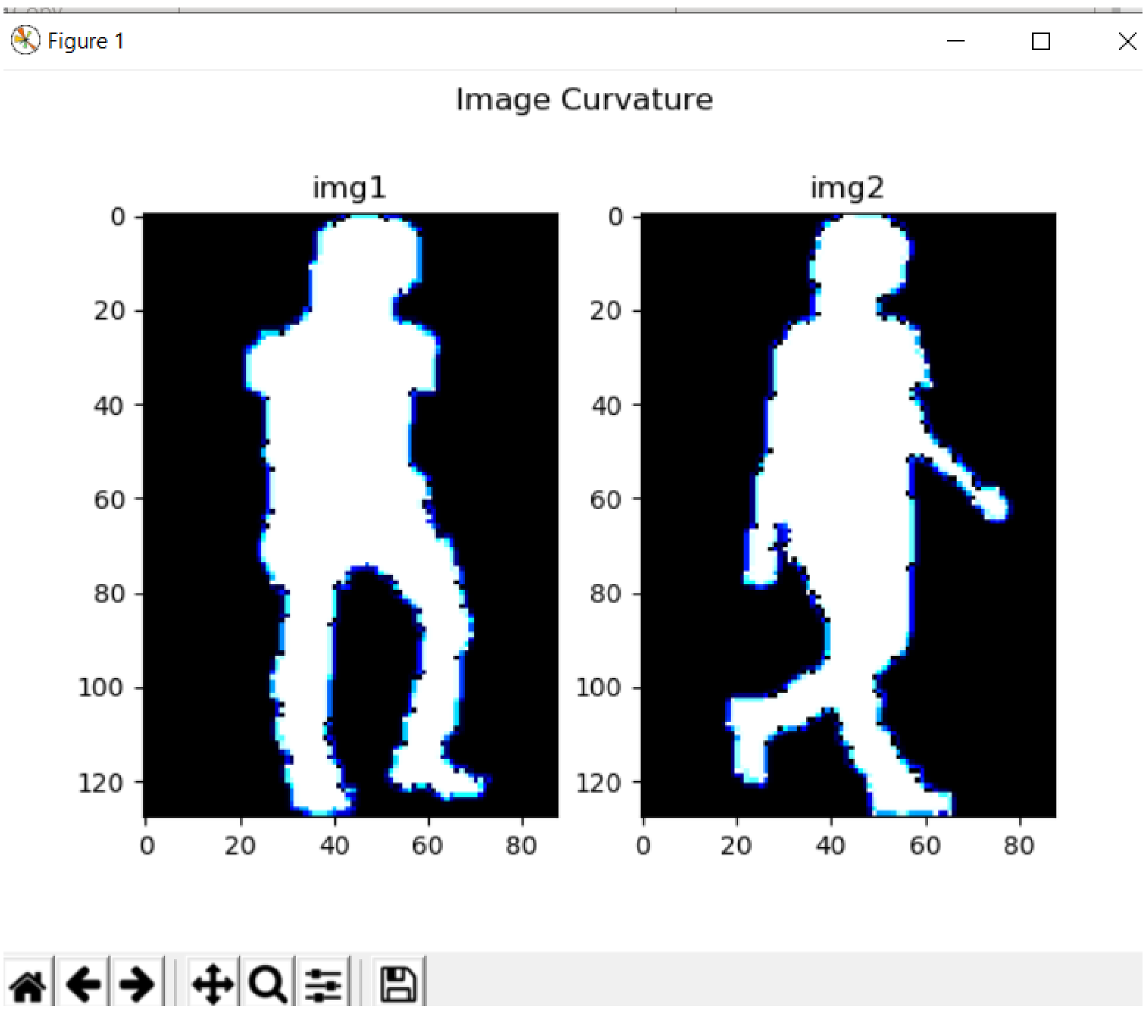
The Curvature formula is: K = 2*(a1*b2 - b1*a2) / (a1*a1 + b1*b1) ^1.5
Trying out different values for K, K=4 seems appropriate. The hotter areas can be observed.

Code:

```
def Curvature(image,k): #Function to compute curvature
    grayimg = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    ret, thresh = cv2.threshold(grayimg, 127, 255,cv2.THRESH_BINARY)
    contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    contr = contours[0]
    cx = []
    cy = []
    for c in contr:
        cx.append(c[0][0])
        cy.append(c[0][1])
    ax = []
    by = []
    for i in range(len(contr)):
        previousindex = i - k
        if(previousindex<0):
            previousindex = len(contr) - i - 1
        nextindex = (i + k) % len(contr)
        ax.append(np.polyfit([previousindex, i, nextindex], [cx[previousindex], cx[i], cx[nextindex]],
2))
        by.append(np.polyfit([previousindex, i, nextindex], [cy[previousindex], cy[i], cy[nextindex]],
2))
    ktan=[]
    for i in range(len(contr)):
        a = ax[i]
        b = by[i]
        ktan.append((((a[1] + 2 * a[0] * i) * 2 * b[0]) - ((b[1] + 2 * b[0] * i) * 2 * a[0])) /
(math.pow(((math.pow((a[1] + 2 * a[0] * i), 2) + math.pow((b[1] + 2 * b[0] * i), 2))), 1.5)))

    ktan = np.multiply(ktan, 255 / max(ktan)).astype(np.uint8)
    for i in range(len(contr)):
        c = contr[i]
```

```
        cx = c[0][0]
        cy = c[0][1]
        grayimg[cy][cx] = ktan[i]
    return grayimg

curve1=Curvature(imglist[0],4)
curve1=cv2.applyColorMap(curve1,11)
curve2=Curvature(imglist[25],4)
curve2=cv2.applyColorMap(curve2,11)
Imageplot(curve1,curve2, "Image Curvature")
```

Output:

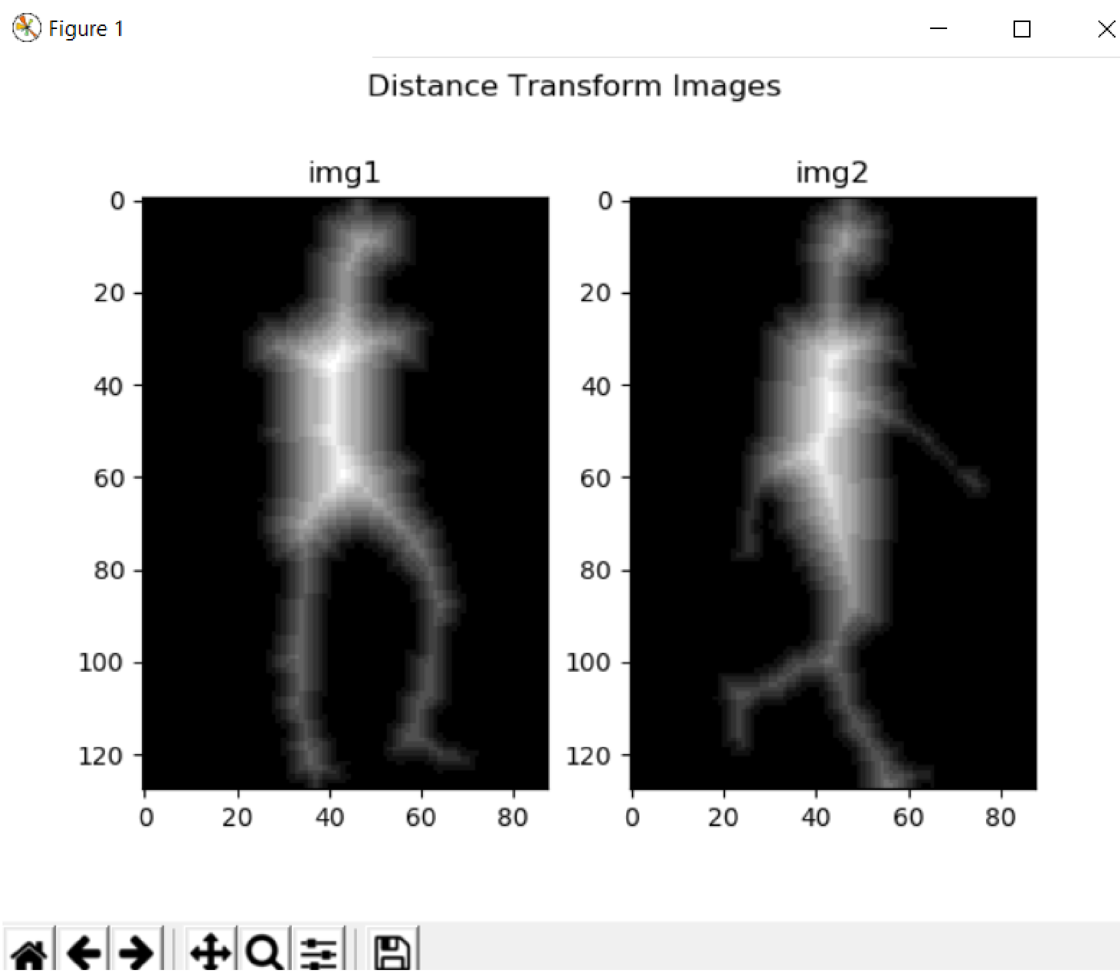


4. Distance Transform is computed using the OpenCV function, for the selected images.

Code:

```
distancetransform = [] #To Compute Distance Transform
for i,index in enumerate(indexes):
   distancetransform.append(np.rint(cv2.distanceTransform(imgselect[i], cv2.DIST_L2, 3)))
Imageplot(distancetransform[0], distancetransform[1],"Distance Transform Images")
```

Output:



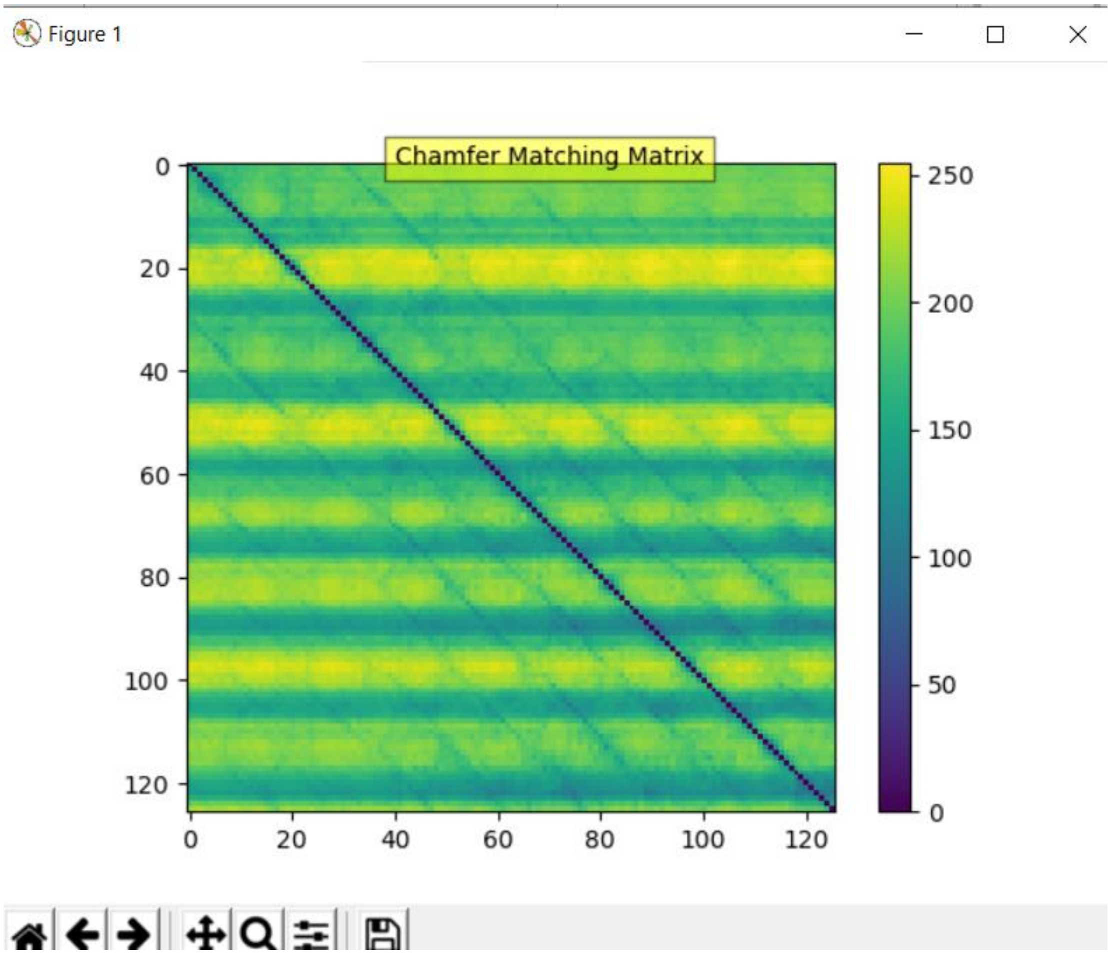
5. Chamfer Matching:

   By computing the scores using image contour as templates and distance transform, Chamfer matching is implemented on all the images.

   Code:

```
def Chamferscore(x,y): #Function to compute Chamfer matching score
   s=np.sum(np.multiply(x,y))
   return s

n=len(imglist)
chamfermatrix=np.zeros((n,n), dtype= np.uint16)
for i in range(n):
   imgtemplate = cv2.Canny(cv2.threshold(cv2.cvtColor(imglist[i], cv2.COLOR_BGR2GRAY), 127,
255, cv2.THRESH_BINARY)[1],120,200)
   for j in range(n):

chmfdistancetransform=cv2.distanceTransform(cv2.Canny(cv2.threshold(cv2.cvtColor(imglist[j],
cv2.COLOR_BGR2GRAY), 127, 255, cv2.THRESH_BINARY)[1], 120, 200), cv2.DIST_L2, 3)
      normdistancetransform = (255-
255*(chmfdistancetransform/np.amax(chmfdistancetransform))).astype(np.uint8)
      chamfermatrix[i][j] = Chamferscore(normdistancetransform, imgtemplate)
chamfermatrix = (255*(chamfermatrix/np.amax(chamfermatrix)))
plt.imshow(chamfermatrix)
plt.colorbar()
plt.text(40,0,"Chamfer Matching Matrix", bbox=dict(facecolor='yellow', alpha=0.5))
          plt.show()
```

Output:



6.

a. The is periodicity present between 2 and 5. Some parts of the images can be analyzed to be similar.
b. There is minute difference in perimeters of convex hull. The area and the perimeter change in a similar periodic manner.
c. The joints and body parts can be analyzed by the points with higher curvature value, hotter points.