Francesco Leone

Exercise 3

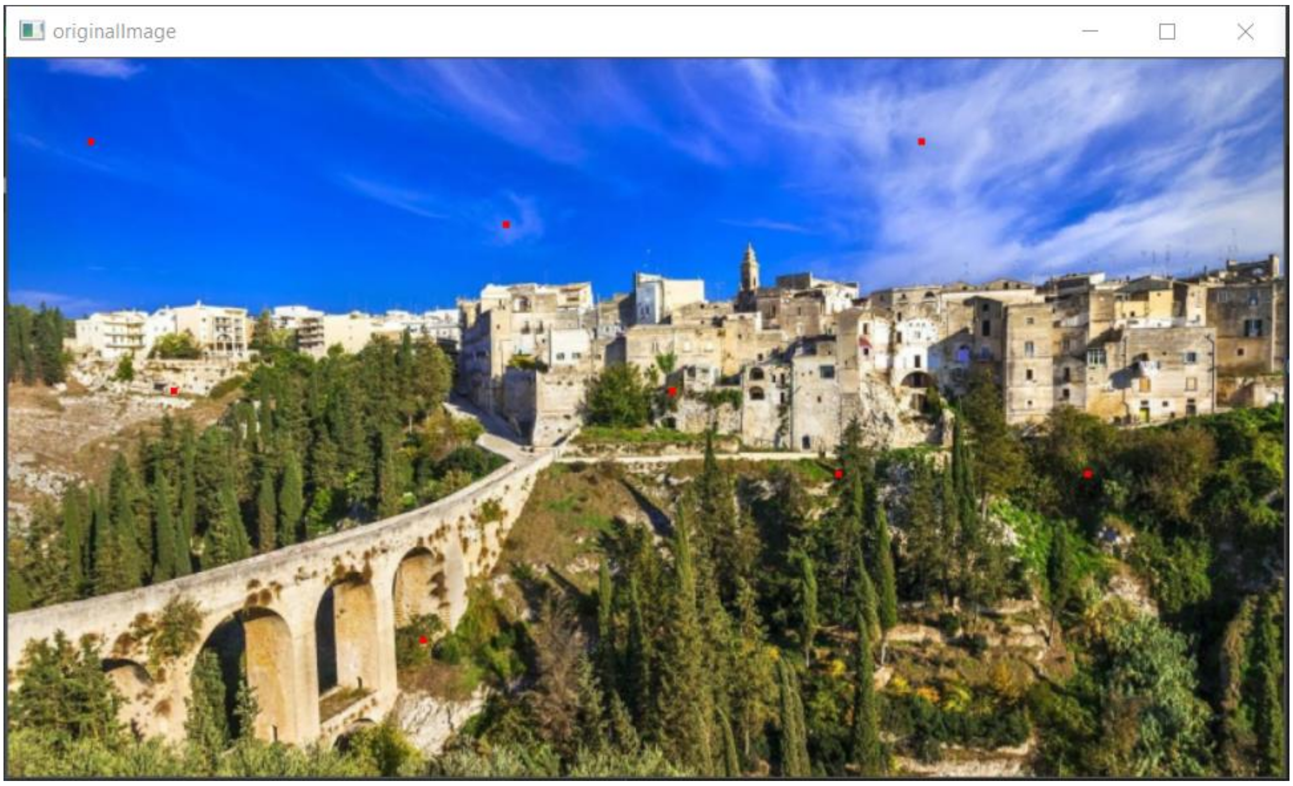
1) Program

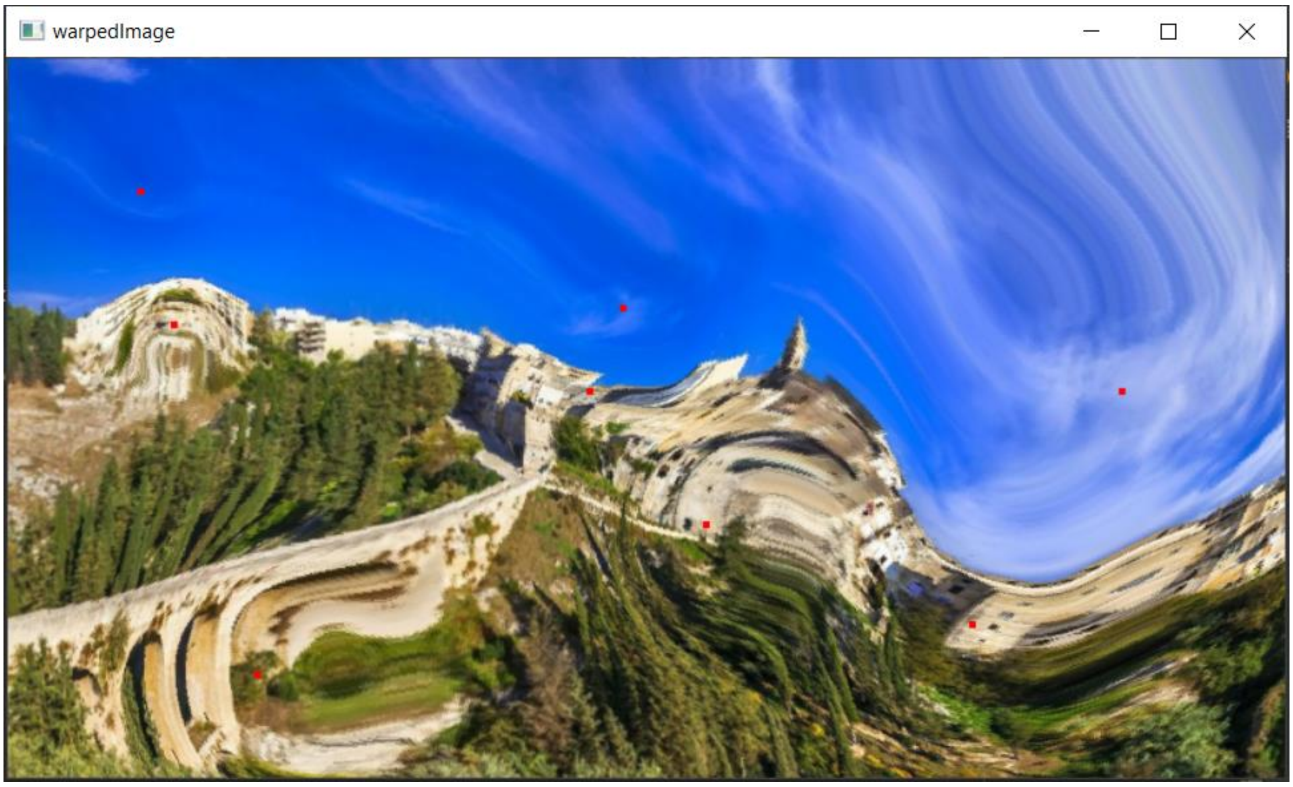
```
import cv2
import numpy as np
import math

def vectorLength(vector):
  return math.sqrt(vector[0] ** 2 + vector[1] ** 2)

def pointsDistance(point1, point2):
  return vectorLength((point1[0] - point2[0],point1[1] - point2[1]))

#if the point is outside the image, reposition it inside
def inside(value, minimum, maximum):
  return max(min(value,maximum),minimum)

#find the color of the warped point (float position) in the original image
def interpolate(image, point):
    height, width, ch = image.shape

    x = point[0]
    y = point[1]
    intX = int(x)
    intY = int(y)
    modX = round(x, 1) - intX
    modY = round(y, 1) - intY

    nextX = 0
    nextY = 0
    if modX == 0 and modY == 0:
        return image[intY,intX]
    if modX > 0.4 :
        nextX = min(intX + 1, width-1)
    else:
        nextX = max(intX - 1, 0)

    if modY > 0.4 :
        nextY = min(intY + 1, height-1)
    else:
        nextY = max(intY - 1, 0)

#these are the 4 nearest neighbors pixels
    a = np.array([intX, intY])
    b = np.array([intX, nextY])
    c = np.array([nextX, nextY])
    d = np.array([nextX, intY])

    distA = pointsDistance(point, a)
    distB = pointsDistance(point, b)
    distC = pointsDistance(point, c)
    distD = pointsDistance(point, d)
```

```
#calculate the influence of each neighbor
    coeff = distA + distB + distC + distD
    coeffA = (distB + distC + distD)/(3*coeff)
    coeffB = (distA + distC + distD) / (3 * coeff)
    coeffC = (distA + distB + distD) / (3 * coeff)
    coeffD = (distA + distB + distC) / (3 * coeff)


    a = np.array([a[0], a[1], coeffA])
    b = np.array([b[0], b[1], coeffB])
    c = np.array([c[0], c[1], coeffC])
    d = np.array([d[0], d[1], coeffD])

#interpolate the color of the pixel
    pixel = np.array([0,0,0])

    for i in (a,b,c,d):
        x = int(i[0])
        y = int(i[1])
        coeff = i[2]
        b, g, r = image[y, x]
        b = b * coeff
        g = g * coeff
        r = r * coeff
        pixel = pixel + (int(b),int(g),int(r))

    return pixel


def warp(image, points):

  height, width, ch = image.shape
  warpedImage = np.zeros((height, width, 3), np.uint8)

  for y in range(0, height):
    for x in range(0, width):

      offset = [0,0]

      for point in points:
        pointPosition = (point[0] + point[2],point[1] + point[3])
        shift_vector = (point[2],point[3])

#Inverse distance weighting  -> empirical
        helper = 1.0 / (3*(pointsDistance((x, y), pointPosition) /
vectorLength(shift_vector)) ** 4 + 1)

        offset[0] -= helper * shift_vector[0]
        offset[1] -= helper * shift_vector[1]

      offset[0] = round(offset[0], 1)
      offset[1] = round(offset[1], 1)
      coords = (inside(x + offset[0], 0, width - 1), inside(y + offset[1], 0,
height - 1))
      warpedImage[y, x] = interpolate(image, coords)
```

```
# highlight the warped points in the original image and in the warped image
  for point in points:
      x = point[0]
      y = point[1]
      for i in range(y - 2, y + 2):
          for j in range(x - 2, x + 2):
              image[i, j] = (0, 0, 255)

      pointPosition = (point[0] + point[2], point[1] + point[3])
      x = pointPosition[0]
      y = pointPosition[1]
      for i in range(y-2, y+2):
          for j in range(x-2, x+2):
              warpedImage[i, j] = (0, 0, 255)

  return warpedImage

#script to warp an image (1280x720)
image = cv2.imread('imageTest.jpg')
width = int(image.shape[1] * 0.6)
height = int(image.shape[0] * 0.6)
dim = (width, height)
image = cv2.resize(image,dim, interpolation = cv2.INTER_AREA)

points = [(50,50,30,30), (100,200,0,-40), (400,200,-50,0), (650,250,-70,90),
(550,50,120,150), (250,350,-100,20), (300,100,70,50), (500,250,-80,30)]

warpedImage = warp(image,points)

cv2.imshow('originalImage',image)
cv2.imshow('warpedImage',warpedImage)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

2) Input and output warped images





The warped points are highlighted in red.

3) Sparse displacement field

A set of 8 tuples (x, y, Δx, Δy), where (x,y) are the coordinates of the point in the original image and (Δx, Δy) are the shift vector.
Tuples: [(50,50,30,30), (100,200,0,-40), (400,200,-50,0), (650,250,-70,90), (550,50,120,150), (250,350,-100,20), (300,100,70,50), (500,250,-80,30)]

(0,0) is the top right position.

4) Dense motion field

The position of each (x, y) pixel of the warped image in the original image is calculate through the following formula:

```
foreach point in the sparse displacement field:
helper = 1.0 / (3*(pointsDistance((x, y), pointPosition) / vectorLength(shift_vector))^4 + 1)
offset[0] -= helper * shift_vector[0]
offset[1] -= helper * shift_vector[1]
x’ = x + offset[0]
y’ = y + offset[0]
```

where (x,y) are the coordinates in the warped image, (x’,y’) are the coordinates in the original image and *helper* is the inverse distance weighting for the shift vector. The factors 3 and 4 in the formula are empirically calculated.

5) Implementation

The implementation is done on the base of the inverse warping. First, the position in the warped image of the selected pixels is calculated with the respective shift vector. Then, the color of the pixels in the warped image is calculate finding their position in the original image: the shift vector of every pixel is inversely proportional to the distance to the nearest warped pixels. This means that if a pixel in the warped picture is far enough from the warped pixels, it will probably remain in the same position as the original image. Once the new position has been found, the color of the pixel is interpolated. The interpolation is done using the 4 nearest neighborhood pixels, each of which contribute to the final value proportionally to how near it is. If the shift vector contributes less than the first decimal to the movement of the pixel, the position is rounded to the integer and interpolation is not needed.

References

Helper formula: https://stackoverflow.com/questions/34884779/whats-a-simple-way-of-warping-an-image-with-a-given-set-of-points