Image Fundamentals

1. Geometric Transformation

```
1.1.You are given two pictures of the same scene, taken at different tim
es. To align the two pictures, you need to
find a mapping function between the two pictures based on some common fe
ature points. Suppose you were
able to extract N (N >= 3) feature points in both images that correspon
d to the same set of object features, with
image coordinates given as (uk, vk) and (xk, yk), k = 1,2, … , N. Also,
suppose you want to use an affine mapping
to approximate the actual unknown mapping function. How would you determ
ine the affine mapping parameters?
```

Affine transformation is a linear mapping method that preserves points, straight lines, and planes. Sets of parallel lines remain parallel after an affine transformation.

The affine transformation technique is typically used to correct for geometric distortions or deformations that occur with non-ideal camera angles.

we have different affine transformations: translation, scale, shear, and rotation. in general we are using image regression wich are classified as bellow: Transformation model: Image registration algorithms can also be classified according to the transformation model used to relate the reference image space with the target image space. The first broad category of transformation models includes linear transformations, which are a combination of translation, rotation, global scaling, shear and perspective components. Linear transformations are global in nature, thus not being able to model local deformations. Usually, perspective components are not needed for registration, so that in this case the linear transformation is an affine one. Frequency-domain methods: algorithms use the properties of the frequency-domain to directly determine shifts between two images. Applying the phase correlation method to a pair of overlapping images produces a third image which contains a single peak. The location of this peak corresponds to the relative translation between the two images. Unlike many spatial-domain algorithms, the phase correlation method is resilient to noise, occlusions, and other defects typical of medical or satellite images. Additionally, the phase correlation uses the fast fourier transform to compute the cross-correlation between the two images, generally resulting in large performance gains. The method can be extended to determine affine rotation and scaling between two images by first converting the images to log-polar coordinates. Due to properties of the fourier transform, the rotation and scaling parameters can be determined in a manner invariant to translation. This single feature makes phase-correlation methods highly attractive vs. typical spatial methods, which must determine rotation, scaling, and translation simultaneously, often at the cost of reduced precision in all three.

```
1.2.Suppose you want to make a panoramic picture of a wide landscape out
of two separately captured pictures with
a certain overlap. Propose and implement an algorithm for stitching up t
he two images to make a panorama. List
the steps involved; Also, discuss and display the results (Test image Ca
r1&2).
```

The process of creating a panoramic image consists of the following steps.

```
.Detect keypoints and descriptors
.Detect a set of matching points that is present in both images (overlap
ping area)
.Apply the RANSAC method to improve the matching process detection
.Apply perspective transformation on one image using the other image as
 a reference frame
.Stitch images together
```

The first step in the process of creating a panorama is to align these two images. For that, we can use the detected features. Basically, we will take features in one image and match them with the features in the other image. However, in order to accomplish our goal of creating a panorama, we need to make sure that there is a common area between these images. The reason for this is that there should be a set of distinctive keypoints detected in this overlapping region. Then, with these keypoints we have an idea of how we should stitch these images together. Then, we need to apply perspective transformation in case that two images are not positioned on the same plane. We will use the first image as a reference frame and warp the second image so that the features in both images are perfectly aligned. This technique is called feature-based image alignment. The final step is to stitch these two images together and to create a panorama image.

1.Detecting distinctive keypoints

necessary libraries are as bellow

In [62]:
```
import sys
import cv2
import numpy as np
from matplotlib import pyplot as plt
```

In [65]:
```
def read_image(path):
    img1 = cv2.imread(path)
    b,g,r = cv2.split(img1)           # get b, g, r
    img1 = cv2.merge([r,g,b])
    return img1
```

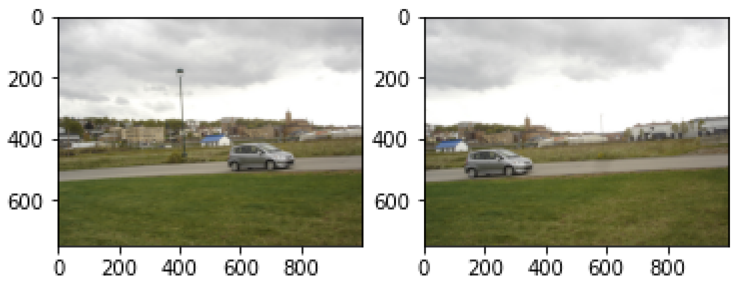
In [66]:
```
def show_images(r,c,inputs):
    plt.figure()
    f, axarr = plt.subplots(r,c)
    for i in range(len(inputs)):
        axarr[i].imshow(inputs[i])
    plt.show()
```

load image

In [67]:
```
img_path1='./1/Car1.jpg'
img_path2='./1/Car2.jpg'
img1 = read_image(img_path1)
img2 = read_image(img_path2)

# img1_gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
# img2_gray = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
show_images(1,2,[img1,img2])
# plt.imshow(img2)
# plt.figure()
# f, axarr = plt.subplots(1,2)
# axarr[0].imshow(img1_gray)
# axarr[1].imshow(img2_gray)
# plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



extract the keypoints

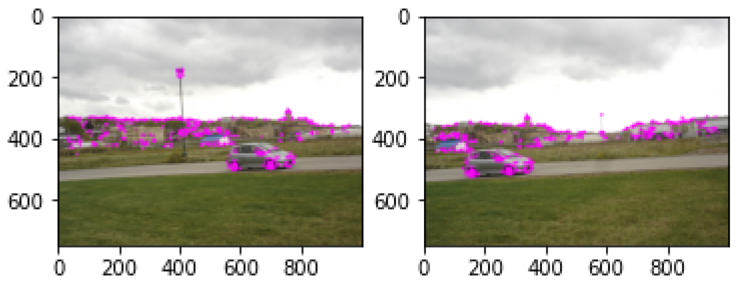
In [68]:
```
orb = cv2.ORB_create(nfeatures=2000)

# Find the key points and descriptors with ORB
keypoints1, descriptors1 = orb.detectAndCompute(img1, None)
keypoints2, descriptors2 = orb.detectAndCompute(img2, None)
```

draw key points in our image

In [69]:
```
output1=cv2.drawKeypoints(img1, keypoints1, None, (255, 0, 255))
output2=cv2.drawKeypoints(img2, keypoints2, None, (255, 0, 255))
show_images(1,2,[output1,output2])
```

```
<Figure size 432x288 with 0 Axes>
```



2.Matching the points between two images

in order to match features, we are going to compare descriptors from the first image with descriptors from the second image. We will also sort matching points by their distance in order to find the closest ones

In [70]:
```
bf = cv2.BFMatcher_create(cv2.NORM_HAMMING)

# Find matching points
matches = bf.knnMatch(descriptors1, descriptors2,k=2)
```

draw_matches() that will be used to match overlapping keypoints

In [71]:
```
def draw_matches(img1, keypoints1, img2, keypoints2, matches):
  r, c = img1.shape[:2]
  r1, c1 = img2.shape[:2]

  # Create a blank image with the size of the first image + second image
  output_img = np.zeros((max([r, r1]), c+c1, 3), dtype='uint8')
  output_img[:r, :c, :] = np.dstack([img1, img1, img1])
  output_img[:r1, c:c+c1, :] = np.dstack([img2, img2, img2])

  # Go over all of the matching points and extract them
  for match in matches:
    img1_idx = match.queryIdx
    img2_idx = match.trainIdx
    (x1, y1) = keypoints1[img1_idx].pt
    (x2, y2) = keypoints2[img2_idx].pt

    # Draw circles on the keypoints
    cv2.circle(output_img, (int(x1),int(y1)), 4, (0, 255, 255), 1)
    cv2.circle(output_img, (int(x2)+c,int(y2)), 4, (0, 255, 255), 1)

    # Connect the same keypoints
    cv2.line(output_img, (int(x1),int(y1)), (int(x2)+c,int(y2)), (0, 255, 255), 

  return output_img
```

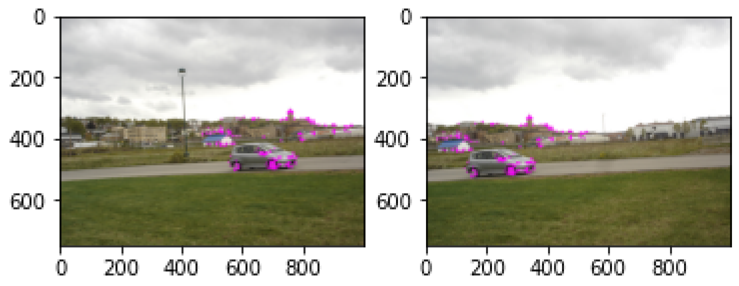
To keep only the strong matches we will use David Lowe’s ratio test. Lowe proposed this ratio test in order to increase the robustness of the SIFT algorithm. Our goal is to get rid of the points that are not distinct enough. Basically, we are discarding these matches where the ratio of the distances to the nearest and the second nearest neighbor is greater than a certain threshold. In this way, we will preserve only good matches.

In [72]:
```
good = []
for m, n in matches:
    if m.distance < 0.6 * n.distance:
        good.append(m)
```

Draw matches

In [73]:
```
output1=cv2.drawKeypoints(img1, [keypoints1[m.queryIdx] for m in good], None, (25
output2=cv2.drawKeypoints(img2, [keypoints2[m.trainIdx] for m in good], None, (25
show_images(1,2,[output1,output2])
```

```
<Figure size 432x288 with 0 Axes>
```



3.Stitching two images together

In [74]:
```
def warpImages(img1, img2, H):

  rows1, cols1 = img1.shape[:2]
  rows2, cols2 = img2.shape[:2]

  list_of_points_1 = np.float32([[0,0], [0, rows1],[cols1, rows1], [cols1, 0]]).r
  temp_points = np.float32([[0,0], [0,rows2], [cols2,rows2], [cols2,0]]).reshape(

  # When we have established a homography we need to warp perspective
  # Change field of view
  list_of_points_2 = cv2.perspectiveTransform(temp_points, H)

  list_of_points = np.concatenate((list_of_points_1,list_of_points_2), axis=0)

  [x_min, y_min] = np.int32(list_of_points.min(axis=0).ravel() - 0.5)
  [x_max, y_max] = np.int32(list_of_points.max(axis=0).ravel() + 0.5)

  translation_dist = [-x_min,-y_min]

  H_translation = np.array([[1, 0, translation_dist[0]], [0, 1, translation_dist[

  output_img = cv2.warpPerspective(img2, H_translation.dot(H), (x_max-x_min, y_ma
  output_img[translation_dist[1]:rows1+translation_dist[1], translation_dist[0]:c

  return output_img
```

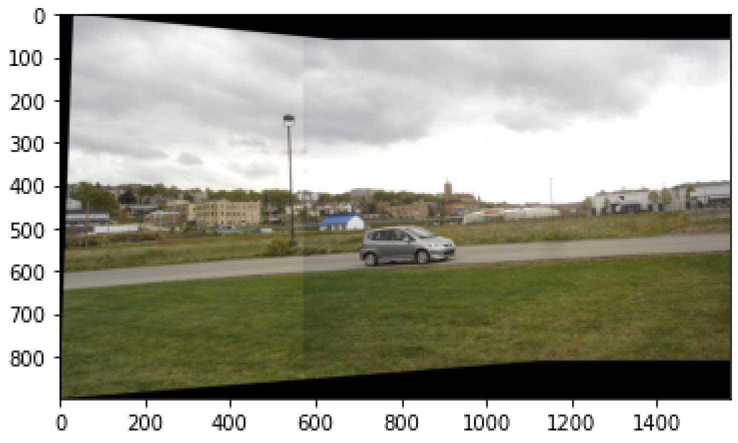
In [75]:
```
MIN_MATCH_COUNT = 15

if len(good) > MIN_MATCH_COUNT:
    # Convert keypoints to an argument for findHomography
    src_pts = np.float32([ keypoints1[m.queryIdx].pt for m in good]).reshape(-1,1
    dst_pts = np.float32([ keypoints2[m.trainIdx].pt for m in good]).reshape(-1,1

    # Establish a homography
    M, _ = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC,5.0)

    result = warpImages(img2, img1, M)

    plt.imshow(result)
```



```
1.1.3. Write a program that implements rotation (with interpolation) of
 an image by angles (45°, 100°, 670°), and
apply it to Barbara image. The rotation center should be the image cente
r
```

import nessary libraries

In [48]:
```
from scipy import ndimage, misc
import matplotlib.pyplot as plt
```

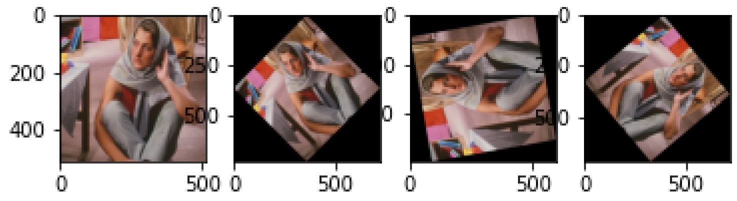
read image

```
In [80]: Barbara='./1/Barbara.bmp'

         barbara_img = read_image(Barbara)
```

```
In [81]: img_45 = ndimage.rotate(barbara_img, 45, reshape=True)
         img_100= ndimage.rotate(barbara_img, 100, reshape=True)
         img_670= ndimage.rotate(barbara_img, 670, reshape=True)
         show_images(1,4,[barbara_img,img_45,img_100,img_670])
```

```
<Figure size 432x288 with 0 Axes>
```



only with numpy

```
In [82]: import numpy as np
         from PIL import Image
         import math
```

In [85]:
```
image = barbara_img             # Load the image
angles=[45,100,670]               # Ask the user to enter the angle of rotation
for angle in angles:
    # Define the most occuring variables
    output_name=str(angle)+"rotated_image.png"
    angle=math.radians(angle)                               #converting degrees t
    cosine=math.cos(angle)
    sine=math.sin(angle)
    height=image.shape[0]                                   #define the height oj
    width=image.shape[1]                                    #define the width of

    # Define the height and width of the new image that is to be formed
    new_height  = round(abs(image.shape[0]*cosine)+abs(image.shape[1]*sine))+1
    new_width  = round(abs(image.shape[1]*cosine)+abs(image.shape[0]*sine))+1

    # define another image variable of dimensions of new_height and new _column f
    output=np.zeros((new_height,new_width,image.shape[2]))

    # Find the centre of the image about which we have to rotate the image
    original_centre_height   = round(((image.shape[0]+1)/2)-1)    #with respect t
    original_centre_width    = round(((image.shape[1]+1)/2)-1)    #with respect t

    # Find the centre of the new image that will be obtained
    new_centre_height= round(((new_height+1)/2)-1)        #with respect to the ne
    new_centre_width= round(((new_width+1)/2)-1)          #with respect to the ne

    for i in range(height):
        for j in range(width):
            #co-ordinates of pixel with respect to the centre of original image
            y=image.shape[0]-1-i-original_centre_height
            x=image.shape[1]-1-j-original_centre_width

            #co-ordinate of pixel with respect to the rotated image
            new_y=round(-x*sine+y*cosine)
            new_x=round(x*cosine+y*sine)

            '''since image will be rotated the centre will change too,
               so to adust to that we will need to change new_x and new_y with re
            new_y=new_centre_height-new_y
            new_x=new_centre_width-new_x

            # adding if check to prevent any errors in the processing
            if 0 <= new_x < new_width and 0 <= new_y < new_height and new_x>=0 ar
                output[new_y,new_x,:]=image[i,j,:]                         #writ

    pil_img=Image.fromarray((output).astype(np.uint8))                    # co
    pil_img.save(output_name)
```

2. Quantization & Interpolation

```
2.1. For two cases as without and with histogram equalization (uniform h
istogram), display the quantized image in
(4, 8, 16, 32, 64, 128) Levels and its histograms. Also, the optimum mea
n square error obtained for each case.
Discuss and report the results for the gray Elaine image. It should be n
oted, you can use rgb2gray, histeq and
immse functions for this problem
```

In [ ]: