IDC410
A course on Image Processing and Machine Learning
(Lecture 06)

Shashikant Dugad,

IISER Mohali

Edge Detection: Recap

- Gradient vector, Single and Double Derivative of Image Matrix
- Robert's 2x2 Edge Detection Operator
- Sobels's 3x3 Edge Detection Operator (c=2)
- Prewitt Edge Detection Operator (c=1)
- Laplacian (∇^2) Edge Detection Operator
- Laplacian-Gaussian ($\nabla^2 G(x,y)$) Edge Detection Operator
- Canny Edge Detection Operator

Canny Edge Detector

Canny Edge Detection Operator

- The Canny edge detection operator ($\nabla G(x,y)$) corresponds to smoothing of an image with a Gaussian filter followed by computation of the gradient
  - $G(x,y)$ is a gaussian filter

- The operator $\nabla G(x,y)$ is <u>NOT</u> rotationally symmetric.

- The operator is symmetric along the edge and antisymmetric perpendicular to the edge i.e. along the line of the gradient

- This means that the operator is sensitive to the edge in the direction of steepest change, but is insensitive to the edge and acts as a smoothing operator in the direction along the edge.

Canny Edge Detection Operator

- **Applying Gaussian filter with spread σ on a image I(I,j)**

$$S[i,j] = G[i,j,: \sigma] \star I[i,j]$$

- **The gradient of the smoothed array S[i, j] can be computed using the 2 x 2 first-difference approximations to produce two arrays $P_x[i,j]$ and $P_y[i,j]$ for the x and y partial derivatives**

$$P_x[i, j] = (S[i, j+1] - S[i, j] + S[i+1, j+1] – S[i+1, j])/2$$

$$P_y[i, j] = (S[i, j] - S[i+1, j] + S[i, j+1] – S[i+1, j+1])/2$$

- **Matrix of magnitude and orientation of the gradient can be computed from the standard formulas**

Canny Edge Detection Operator

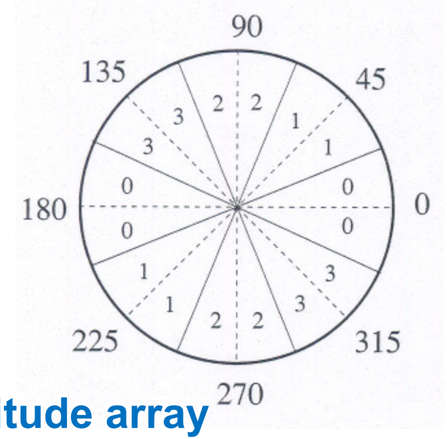
- **The magnitude and orientation of the gradient can be computed from the standard formulas for rectangular-to-polar conversion:**

$$M[i,j] = \sqrt{P_x^2[i,j] + P_y^2[i,j]} \quad \text{and} \quad \theta[i,j] = \tan^{-1}\frac{P_y[i,j]}{P_x[i,j]}$$

- **To identify real edges in the image, the broad ridges in the magnitude M[i,j] array must be thinned so that only the magnitudes at the points of greatest local change remain. This process is called *non-maxima suppression*, (NMS) which in this case results in thinned edges.**

- **NMS thins the ridges by suppressing all values along the line of the gradient that are not peak values of a ridge.**

Non-maxima suppression



- Map the angle of the gradient θ[i,j] to one of the four sectors shown:

$$\Psi[i,j] = \mathrm{Sector}(\theta[i,j])$$

- The algorithm passes a 3x3 neighbourhood across the magnitude array M[i,j].

- If the magnitude of element in 3x3 neighbourhood having the same sector number Ψ[i,j] (broadly ensuring same direction of gradient) is higher than M[i,j] then M[i,j] is set to ZERO

- This process thins the broad ridges of gradient magnitude in M[i, j] into ridges that are only one pixel wide.

NMS → Thresholding

- **Matrix obtained through a process of NMS is referred as matrix of ridges**

- **The threshold is applied on NMS matrix**
  - **Lower threshold may result in false positive**
  - **Higher threshold may result in false negative**

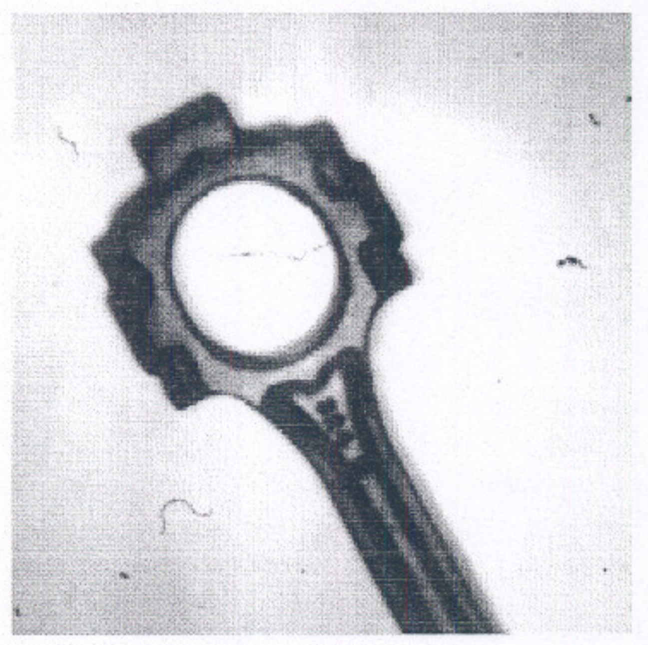
- **Double thresholding ($\tau_2 \approx 2\tau_1$) approach is used to ascertain the location of edges to produce two threshold edge images T1[i,j] and T2[i,j]. Image T2 may contain fewer false edges due to higher threshold but , it may have gaps in the contours (too many false negatives) that are filled using neighbourhood elements in edge image T1**

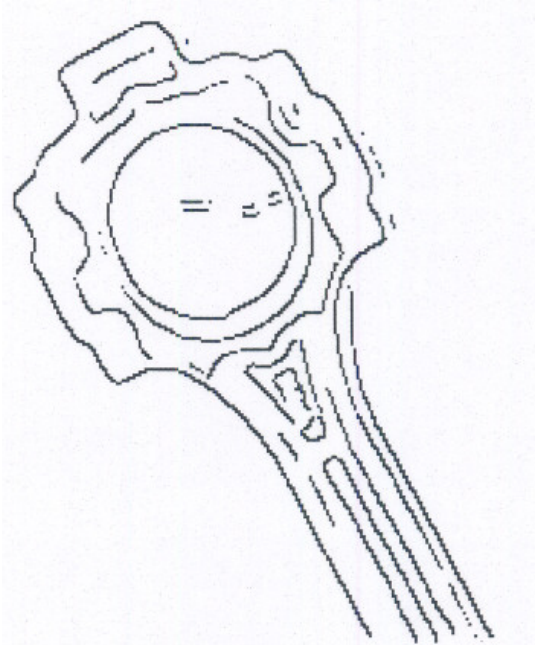
Canny Edge Detection Summary

Steps to follow for implentation:

1. **Smooth the image with a Gaussian filter.**
2. **Compute the gradient magnitude and orientation using finite-difference approximations for the partial derivatives.**
3. **Apply non-maxima suppression to the gradient magnitude.**
4. **Use the double thresholding algorithm to detect and link edges.**

Results of Canny Edge Detection





Contours (Linear Segments and Circles)

Contours: Segments and Circles

- ***Edge list:*** *It is an ordered set of edge points or fragments*

- ***Contour:*** *It is the curve that has been used to represent the edge list*

- ***Boundary:*** *It is the closed contour that surrounds a region*

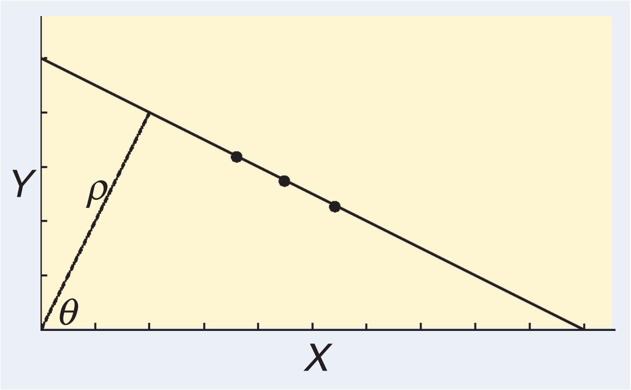
- **Geometry of Curves: Planar curves can be represented in three different ways: a) the explicit form $y = f(x)$, b) the implicit form $f(x,y) = 0$ and c) the parametric form $(x(u),y(u))$. The explicit form is rarely used since in CV a curve can be many-to-many. Parametric form is preferred to represent digital curves in CV**

Curve Fitting

- **Broadly CV handles four types of curves namely:**
    1. **Line segments**
    2. **Circular arcs**
    3. **Conic sections**
    4. **Cubic splines**
- **We shall cover identification of *line segments* and *circular arcs* using Hough Transforms (HT).**
- **The HT provides the technique to find imperfect instances of objects within a certain class of shapes by a voting procedure.**
- **The voting procedure is carried out in a *parameter space* of a given class of shape (line segment or circular arc etc.) from which object candidates are obtained as local maxima in a so-called accumulator space**

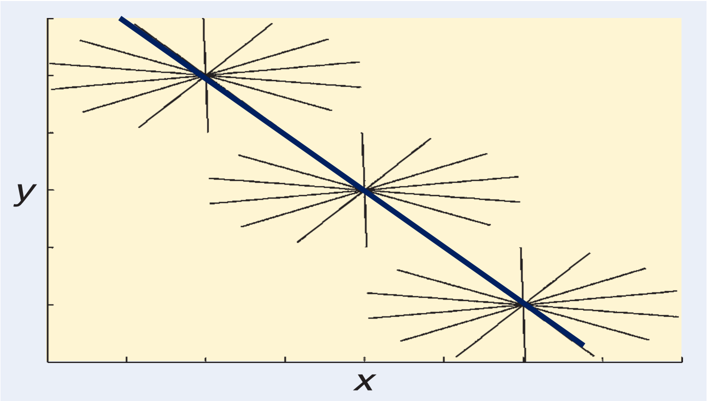
Hough Transform for line segment

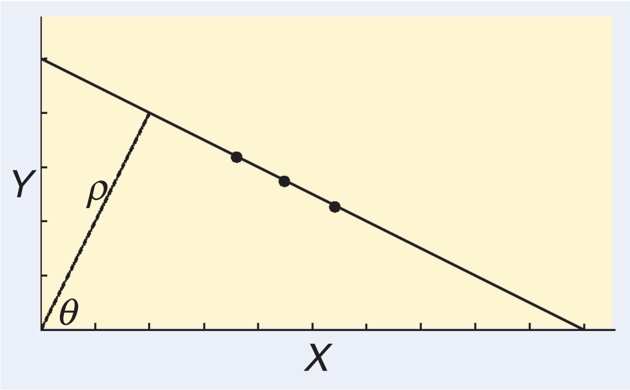
- Equation of a line in image x-y plane: $y_i = mx_i + c$. (x_i, y_i) is variable; and (c, m) is constant)
- Representation of a point in a line Parameter space c-m: (c, m) is variable and (x_i, y_i) is constant)
    - However, m and c can go to $\pm\infty$
    - Hesse normal form of line equation is preferred: $\rho = x_i\text{Cos}(\theta) + y_i\text{Sin}(\theta)$



Hough Transform for a line segment

- Equation of a line in image x-y plane: $y_i = mx_i + c$. (x_i, y_i) is variable; and (c, m) is constant). Infinite lines can be drawn from a given point
  - How to find a common line that goes through all 3 points?
- Representation of a point in a line Parameter space c-m: (c, m) is variable and (x_i, y_i) is constant)
  - However, m can go to $\pm\infty$
  - Hesse normal form of line equation is preferred: $\rho = x_i\cos(\theta) + y_i\sin(\theta)$



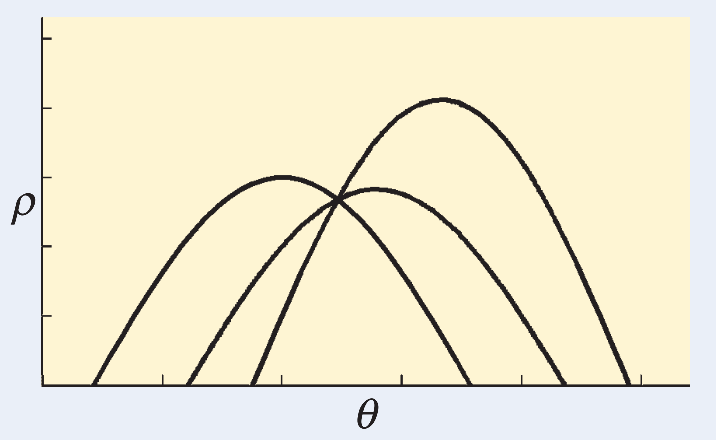


Source: Peter Hart, IEEE SIGNAL PROCESSING MAGAZINE [20] NOVEMBER 2009

https://en.wikipedia.org/wiki/Hough_transform

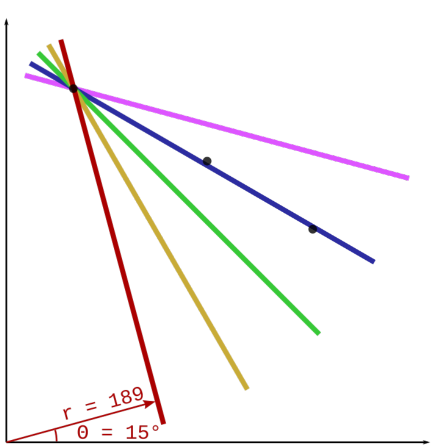
Hough Transform for a line segment

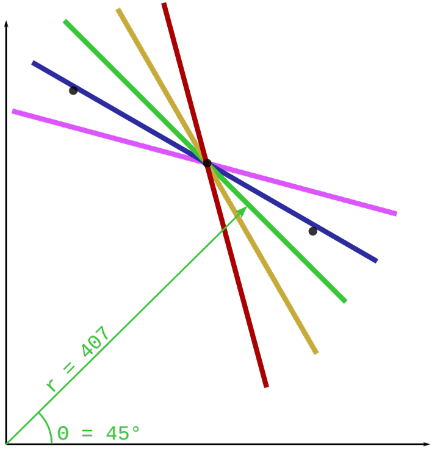
- Given a *single point* (x_i, y_i) in the image (x-y) plane, the set of *all* straight lines going through that point corresponds to a sinusoidal curve in the $\rho - \theta$ plane, that is unique to that point ($\rho > 0$ and $0 < \theta < 2\pi$)
- The intersection of the sinusoids corresponds to the line in the x–y space passing through the three points.

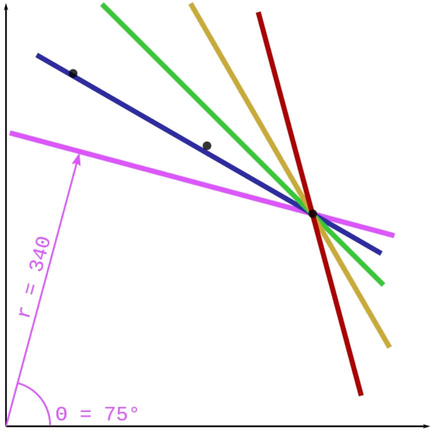


- Thus, the problem of detecting collinear points gets translated to the problem of finding concurrent curves

Example: (https://en.wikipedia.org/wiki/Hough_transform)







θ	r
15	189.0
30	282.0
45	355.7
60	407.3
75	429.4

θ	r
15	318.5
30	376.8
45	407.3
60	409.8
75	385.3

θ	r
15	419.0
30	443.6
45	438.4
60	402.9
75	340.1

Hough Transform for a circle

Source: https://en.wikipedia.org/wiki/Circle_Hough_Transform

- In a two-dimensional space, a circle can be described by:

$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

- where (a, b) is the center of the circle, and r is the radius. If a 2D point (x_i, y_i) is fixed, then the parameters a, b, r can be found accordingly. The parameter space would be three dimensional, (a, b, r). All the parameters that satisfy (x_i, y_i) would lie on the surface of an inverted right-angled cone whose apex is at (x_i, y_i, 0), i.e. ($a=x_i$, $b=y_i$ and $r=0$) In the 3D space, the circle parameters can be identified by the intersection of many conic surfaces that are defined by points on the 2D circle.

- Create the accumulator space, for each pixel in the image and find the parameters of a circle having the maximum score

Detection of known Shapes

- There are other shapes like triangle, rectangles etc needs to be identified
- If there exact shape with dimensions are known, then you can create a filter with that shape dimension
- Apply it over the image and obtain the convoluted image to see the location of the shape if it exists!
- It’s a crude approach!!

Geometric Transformation

Geometric Transformation

- **Geometric transformations (or operations) are extensively used in image processing:**
  - **They allow us to bring multiple images into the same frame of reference so that they can be combined or compared**
  - **They can be used to eliminate distortion in order to create images with evenly spaced pixels.**
  - **Geometric trans formations can also simplify further processing**

- **Given a distorted (original) image f (i, j) and a corrected (transformed) image f ′(i′, j′); we can model the geometric transformation between their coordinates of distorted and corrected image as:**

$$i' = T_{i'}(i,j) \quad and \quad j' = T_{j'}(i,j)$$

Geometric Transformation

- Given the coordinates (i, j) in distorted (original) image; coordinates (i′, j′) in the corrected image can be obtained using the geometric transformation functions; $T_{i'}(i,j)\ and T_{j'}(i,j)$

- Transformation functions may be defined in advance

- Or it can be determined with a priori knowledge of corelation between few points in the distorted image and corresponding points in the corrected image

- ***Affine transformation*** approach is used for geometric transformation to obtained the corrected image using distorted image as input

Affine Transformation

- **Many features of geometric transformation of an image, such as translation, rotation, de-skewing etc. can be obtained using *Affine Transformation* which is defined as follows:**

$$\begin{bmatrix} i' \\ j' \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix} \begin{bmatrix} i \\ j \\ 1 \end{bmatrix}$$

$i' = a_{00}i + a_{01}j + a_{02}$ **and** $j' = a_{10}i + a_{11}j + a_{12}$

- **Translation by m along the horizontal axis and n along the vertical axis:**

$$\begin{bmatrix} i' \\ j' \end{bmatrix} = \begin{bmatrix} 1 & 0 & m \\ 0 & 1 & n \end{bmatrix} \begin{bmatrix} i \\ j \\ 1 \end{bmatrix} . \qquad i' = i + m. \quad j' = j + n$$

Affine Transformation

- **Change of Scale (Expand/Shrink):** Change of scale **a** in the horizontal axis and **b** in the vertical axis. This is often required in order to normalise the size of objects in images:

$$\begin{bmatrix} i' \\ j' \end{bmatrix} = \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \end{bmatrix} \begin{bmatrix} i \\ j \\ 1 \end{bmatrix} . \qquad i' = ai. \quad j' = bj$$

- **Rotation:** Rotation by angle ϕ about the origin. This is sometimes required to align an image with the axes to simplify further processing

$$\begin{bmatrix} i' \\ j' \end{bmatrix} = \begin{bmatrix} cos(\phi) & sin(\phi) & 0 \\ -sin(\phi) & cos(\phi) & 0 \end{bmatrix} \begin{bmatrix} i \\ j \\ 1 \end{bmatrix}$$

Affine Transformation

- **Skewing**: Skewing by angle ϕ if often needed to remove nonlinear viewing effects such as those generated by a line scan camera:

$$\begin{bmatrix} i' \\ j' \end{bmatrix} = \begin{bmatrix} 1 & tan(\phi) & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} i \\ j \\ 1 \end{bmatrix}$$

- **Panoramic distortion:** Panoramic distortion is in effect an incorrect aspect ratio. It appears in line scanners when the mirror rotates at an incorrect speed

$$\begin{bmatrix} i' \\ j' \end{bmatrix} = \begin{bmatrix} a & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} i \\ j \\ 1 \end{bmatrix}$$

Unknown Affine Transformations

- Often, the coefficients transformation (a_{lm}) are not known in advance.
- These coefficients can be determined if we have knowledge of at least three well separated points in original and corrected image, by solving following equations:

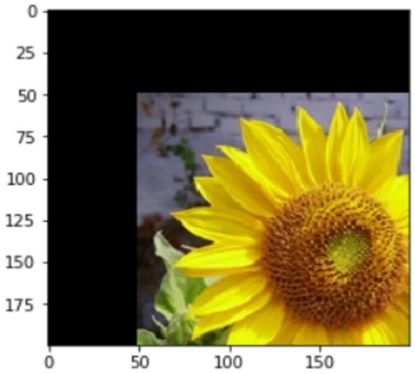
$$\begin{bmatrix} i_1 \\ j_1 \\ i_2 \\ j_2 \\ i_3 \\ j_3 \end{bmatrix} = \begin{bmatrix} i'_1 & j'_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & i'_1 & j'_1 & 1 \\ i'_2 & j'_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & i'_2 & j'_2 & 1 \\ i'_3 & j'_3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & i'_3 & j'_3 & 1 \end{bmatrix} \begin{bmatrix} a_{00} \\ a_{01} \\ a_{02} \\ a_{10} \\ a_{11} \\ a_{12} \end{bmatrix}$$

Top right: Computed from correspondences for the top left, top right and bottom left corners of the license plate,

Bottom right: Computed from correspondences for all four corners of the license plate

Image Transformation

Input Image



Translated Output Image

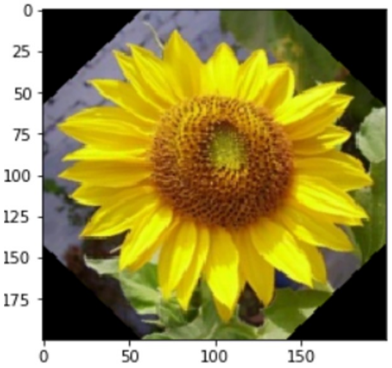


Image Rotated by 45 degrees

Panorama Stitching

Step 1 - Detect the key points in both the input images.

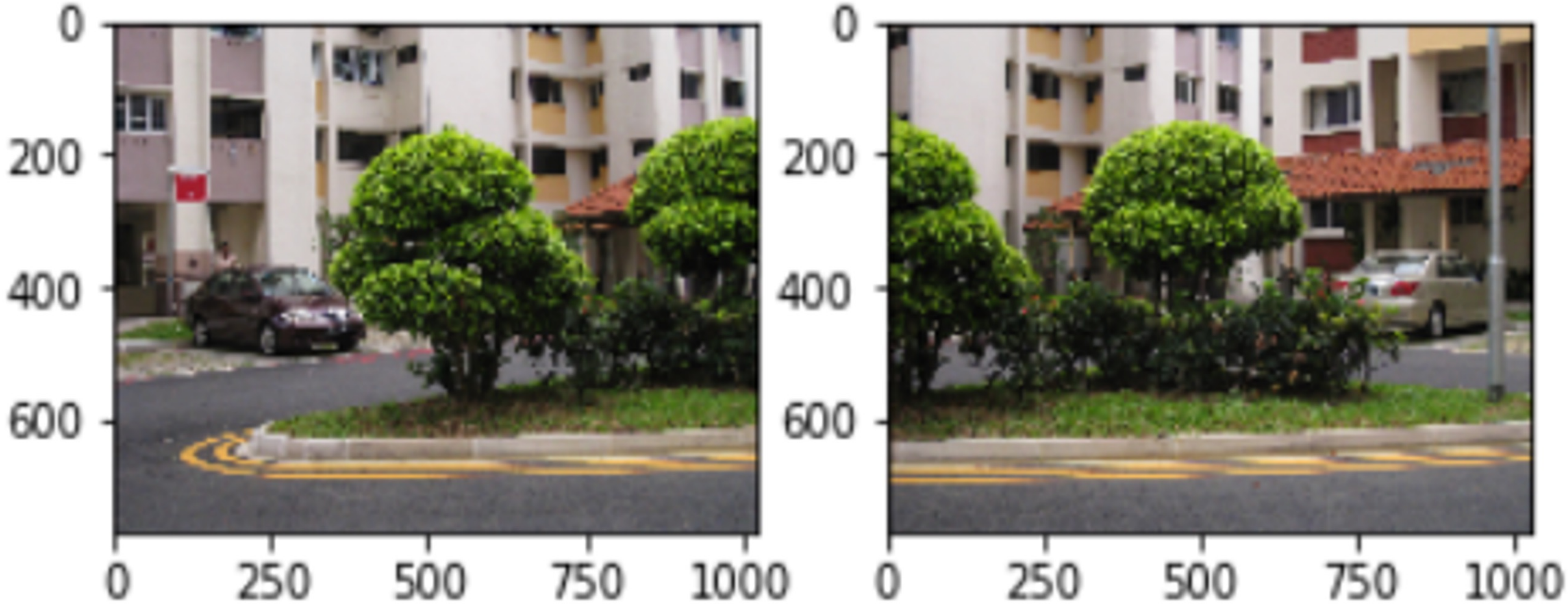
Step 2 - Match Keypoints between the two images to be stitched.

Step 3 - Stitch the two images if they have matching key points otherwise the images cannot be stitched.

Panorama Stitching

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

# read the two input image, convert it into RGB scale and resize for further operations
dim=(1024,768)
left=cv2.imread('img1.jpg',cv2.IMREAD_COLOR)
left=cv2.resize(left,dim,interpolation = cv2.INTER_AREA)    #ReSize to (1024,768)
left1 = cv2.cvtColor(left,cv2.COLOR_BGR2RGB)

right=cv2.imread('img2.jpg',cv2.IMREAD_COLOR)
right=cv2.resize(right,dim,interpolation = cv2.INTER_AREA) #ReSize to (1024,768)
right1 = cv2.cvtColor(right,cv2.COLOR_BGR2RGB)

# for plotting the two input images side by side
f, axarr = plt.subplots(1,2)
axarr[0].imshow(left1)
axarr[1].imshow(right1)

```

Panorama Stitching



Panorama Stitching

```python
# create a list which will contain the two input images
images=[]
images.append(left)
images.append(right)

#stitcher = cv2.createStitcher()
stitcher = cv2.Stitcher.create()

# ret contains True if pano contains a valid output else False
ret,pano = stitcher.stitch(images)

if ret==cv2.STITCHER_OK:
    pano = cv2.cvtColor(pano,cv2.COLOR_BGR2RGB)
    plt.title('Panorama Image')
    plt.imshow(pano)

else:
    print("Given images cannot be stitched!")
```

Panorama Stitching