Chapter 4

Edge Detection and Corner Detection in Images and Videos

This chapter deals with edge detection and corner detection in images and videos using Python 3.8.13. I hope you will like it.

Contents:

1. Edge Detection in Images

Edge detection is an image-processing technique, which is used to identify the boundaries (edges) of objects, or regions within an image. Edges are among the most important features associated with images. We come to know of the underlying structure of an image through its edges. Computer vision processing pipelines therefore extensively use edge detection in applications.

How are Edges Detected?

Edges are characterized by sudden changes in pixel intensity. To detect edges, we need to go looking for such changes in the neighboring pixels. We will use some of edge-detection algorithms available in OpenCV which are:

- Prewitt Edge Detection
- Sobel Edge Detection
- Laplacian Edge Detection
- Canny Edge Detection

1.1 Prewitt Edge Detection

This method is a commonly used edge detector mostly to detect the horizontal and vertical edges in images. The following are the Prewitt edge detection filters:

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \qquad G_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

Prewitt filter for vetical edge detection **Prewitt filter for horizontal edge detection**

Demonstration

For More Information Please Visit The Following Links:

https://www.mygreatlearning.com/blog/introduction-to-edge-detection/ (https://www.mygreatlearning.com/blog/introduction-to-edge-detection/)

https://gist.github.com/rahit/c078cabc0a48f2570028bff397a9e154 (https://gist.github.com/rahit/c078cabc0a48f2570028bff397a9e154)

https://www.analyticsvidhya.com/blog/2021/03/edge-detection-extracting-the-edges-from-an-image/ (https://www.analyticsvidhya.com/blog/2021/03/edge-detection-extracting-the-edges-from-an-image/)

https://www.bogotobogo.com/python/OpenCV_Python/python_opencv3_Image_Gradient_Sobel_Lap (https://www.bogotobogo.com/python/OpenCV_Python/python_opencv3_Image_Gradient_Sobel_La

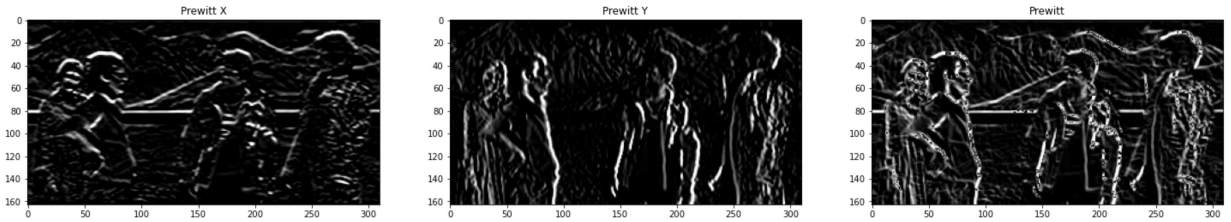
https://learnopencv.com/edge-detection-using-opencv/ (https://learnopencv.com/edge-detection-using-opencv/)

In [1]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
image = cv2.imread('3Idiots.png',0)
plt.figure(figsize=(15,5))
plt.title('Original Image')
plt.imshow(image,cmap='gray')
plt.axis('off')
plt.show()
```

In [3]:
```
img_blur = cv2.GaussianBlur(image, (3,3), 0)
plt.figure(figsize=(15,5))
plt.title('Image After Applying Gaussian Filter')
plt.imshow(img_blur,cmap='gray')
plt.axis('off')
plt.show()
```

In [4]:
```
kernelx = np.array([[1,1,1],[0,0,0],[-1,-1,-1]])
kernely = np.array([[-1,0,1],[-1,0,1],[-1,0,1]])
img_prewittx = cv2.filter2D(img_blur,-1,kernelx)
img_prewitty = cv2.filter2D(img_blur,-1,kernely)
plt.figure(figsize=(25,25))
plt.subplot(131)
plt.title("Prewitt X")
plt.imshow(img_prewittx,cmap='gray')
plt.subplot(132)
plt.title("Prewitt Y")
plt.imshow(img_prewitty,cmap='gray')
plt.subplot(133)
plt.title("Prewitt")
plt.imshow(img_prewittx + img_prewitty,cmap='gray')
plt.show()
```



1.2 Sobel Edge Detection:

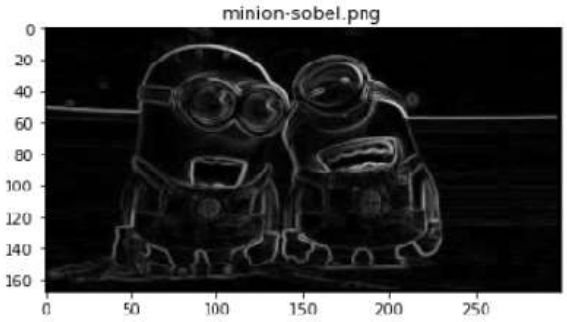
This uses a filter that gives more emphasis to the centre of the filter. It is one of the most commonly used edge detectors and helps reduce noise and provides differentiating, giving edge response simultaneously. The following are the filters used in this method:

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Sobel filter for vertical edge detection **Sobel filter for horizontal edge detection**

The following shows the before and after images of applying Sobel edge detection:

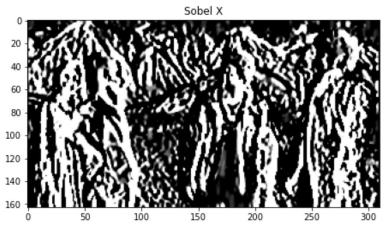
original minion image

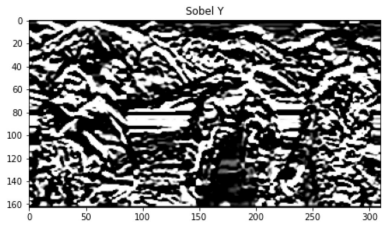


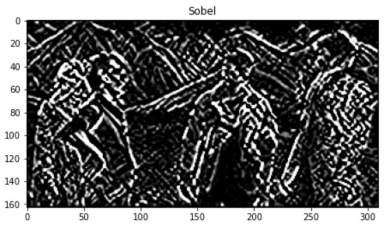
minion image after applying sobel edge detector

Demonstration

In [5]:
```
img_sobelx = cv2.Sobel(img_blur,cv2.CV_8U,1,0,ksize=5)
img_sobely = cv2.Sobel(img_blur,cv2.CV_8U,0,1,ksize=5)
img_sobel = cv2.Sobel(img_blur,cv2.CV_8U,1,1,ksize=5)
plt.figure(figsize=(25,25))
plt.subplot(131)
plt.title("Sobel X")
plt.imshow(img_sobelx,cmap='gray')
plt.subplot(132)
plt.title("Sobel Y")
plt.imshow(img_sobely,cmap='gray')
plt.subplot(133)
plt.title("Sobel")
plt.imshow(img_sobel,cmap='gray')
plt.show()
```







1.3 Laplacian Edge Detection

The Laplacian edge detectors vary from the previously discussed edge detectors. This method uses only one filter (also called a kernel). In a single pass, Laplacian detection performs second-order derivatives and hence are sensitive to noise. To avoid this sensitivity to noise, before applying this method, Gaussian smoothing is performed on the image.

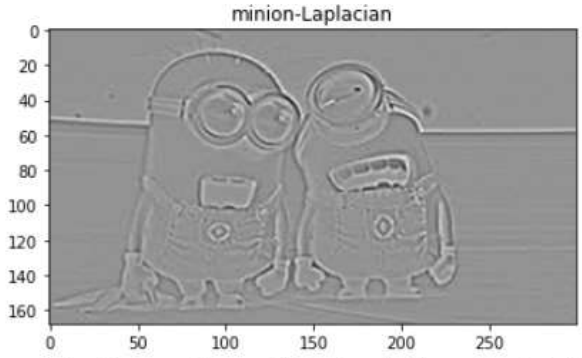
−1	−1	−1
−1	8	−1
−1	−1	−1

0	−1	0
−1	4	−1
0	−1	0

Common Laplacian edge detection filters

The above are some of the commonly used Laplacian edge detector filters that are small in size. The following shows the original minion image and the final image after applying Gaussian smoothing (GaussianBlur() method of cv2) followed by Laplacian detection:

original minion image



minion image after applying Laplacian edge detection

Demonstration

In [6]:
```
laplacian = cv2.Laplacian(img_blur,cv2.CV_8U)
plt.figure(figsize=(15,5))
plt.title('Lapacian')
plt.imshow(laplacian,cmap='gray')
plt.axis('off')
plt.show()
```

1.4 Canny Edge Detection

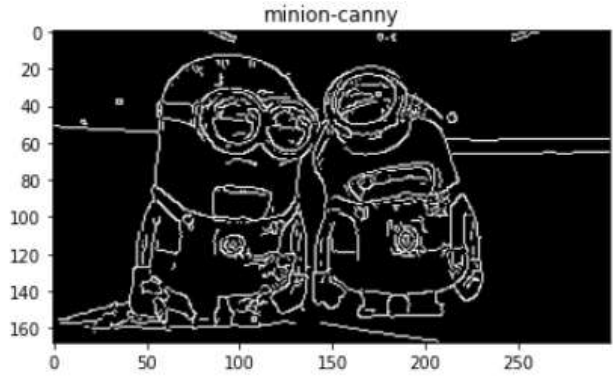
This is the most commonly used highly effective and complex compared to many other methods. It is a multi-stage algorithm used to detect/identify a wide range of edges.

- Convert the image to grayscale
- Reduce noise – as the edge detection that using derivatives is sensitive to noise, we reduce it.
- Calculate the gradient – helps identify the edge intensity and direction.
- Non-maximum suppression – to thin the edges of the image.
- Double threshold – to identify the strong, weak and irrelevant pixels in the images.

- Hysteresis edge tracking – helps convert the weak pixels into strong ones only if they have a strong pixel around them.

The following are the original minion image and the image after applying this method.

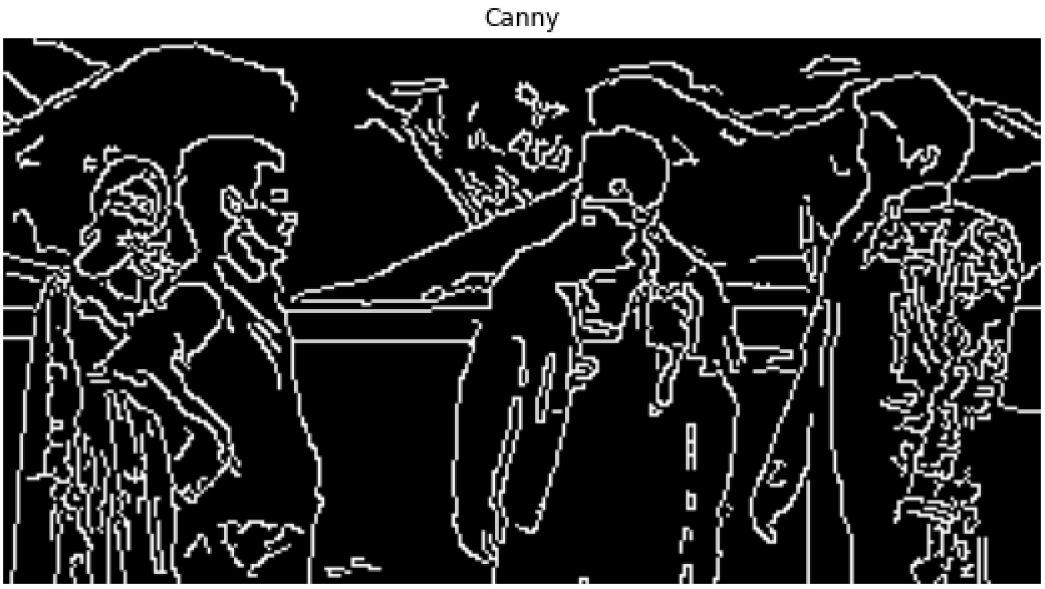
original minion image



minion image after applying canny edge detection

Demonstration

In [7]:
```
canny = cv2.Canny(img_blur,100,200)
plt.figure(figsize=(15,5))
plt.title('Canny')
plt.imshow(canny,cmap='gray')
plt.axis('off')
plt.show()
```



Drawbacks of Applying Edge Computation

1. **Size of output will be shrunk.** If you notice in the above example with an input of 66 *image after applying 33 filter, the output image is only 44. Usually, the formula is if the size of the input image is nn and the filter size is rr, the output image size will be (n-r+1)*(n-r+1).

2. **Loss of a lot of valuable information, especially from the edges of the input image.** As the output image size is much reduced than the original image used as input (as discussed above), the information towards the edges of the input image is lost as we don't iterate multiple times using the filter on the input images' outer edges (unlike the middle of the input image).

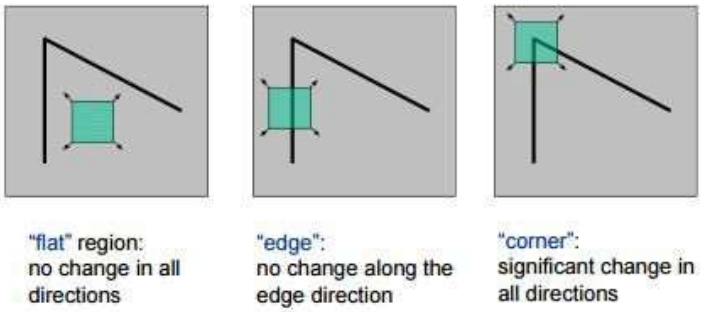
Techniques to Overcome the Drawbacks of Edge Computation

To prevent the loss of such valuable information by image shrinkage, we usually use "padding" the input image before applying detection to avoid losing the valuable information in the input images.

2. Corner Detection

Corner detection is an approach used within computer vision systems to extract certain kinds of features and infer the contents of an image. Corner detection is frequently used in motion detection, image registration, video tracking, image mosaicing, panorama stitching, 3D reconstruction and object recognition. Corner detection overlaps with the topic of interest point detection.

A corner is a point whose local neighborhood stands in two dominant and different edge directions. In other words, a corner can be interpreted as the junction of two edges, where an edge is a sudden change in image brightness. Corners are the important features in the image, and they are generally termed as interest points which are invariant to translation, rotation, and illumination.



So let's understand why corners are considered better features or good for patch mapping. In the above figure, if we take the flat region then no gradient change is observed in any direction. Similarly, in the edge region, no gradient change is observed along the edge direction. So both flat region and edge region are bad for patch matching since they not very distinctive (there are many similar patches in along edge in edge region). While in corner region we observe a significant gradient change in all direction. Due this corners are considered good for patch matching(shifting the window in any direction yield a large change in appearance) and generally more stable over the change of viewpoint.

There are different kinds of Corner Detection algorithms that are listed below:

- The Harris & Stephens / Shi–Tomasi corner detection algorithms
- The Förstner corner detector
- The multi-scale Harris operator
- The level curve curvature approach

- Laplacian of Gaussian, differences of Gaussians and determinant of the Hessian scale-space interest points
- Scale-space interest points based on the Lindeberg Hessian feature strength measures
- Affine-adapted interest point operators
- The Wang and Brady corner detection algorithm
- The SUSAN corner detector
- The Trajkovic and Hedley corner detector
- AST-based feature detectors
- Automatic synthesis of detectors
- Spatio-temporal interest point detectors

In this section we are going to discuss about The Harris & Stephens / Shi–Tomasi corner detection algorithms.

For Further Information Please Visit The Given Links:

https://pythonprogramming.net/corner-detection-python-opencv-tutorial/ (https://pythonprogramming.net/corner-detection-python-opencv-tutorial/)

https://docs.opencv.org/4.x/dc/d0d/tutorial_py_features_harris.html (https://docs.opencv.org/4.x/dc/d0d/tutorial_py_features_harris.html)

https://www.geeksforgeeks.org/python-corner-detection-with-harris-corner-detection-method-using-opencv/ (https://www.geeksforgeeks.org/python-corner-detection-with-harris-corner-detection-method-using-opencv/)

https://www.geeksforgeeks.org/python-corner-detection-with-shi-tomasi-corner-detection-method-using-opencv/?ref=lbp (https://www.geeksforgeeks.org/python-corner-detection-with-shi-tomasi-corner-detection-method-using-opencv/?ref=lbp)

https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_fast/py_fast.html (https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_fast/py_fast.html)

https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_shi_tomasi/py_shi_tomasi.html (https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_shi_tomasi/py_shi_tomasi.html)

https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_features_harris/py_features_harris.ht (https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_features_harris/py_features_harris.ht

https://www.southampton.ac.uk/~msn/book/new_demo/corners/#:~:text=Corner%20detection%20wo (https://www.southampton.ac.uk/~msn/book/new_demo/corners/#:~:text=Corner%20detection%20w

https://en.wikipedia.org/wiki/Corner_detection (https://en.wikipedia.org/wiki/Corner_detection)

https://medium.com/data-breach/introduction-to-harris-corner-detector-32a88850b3f6 (https://medium.com/data-breach/introduction-to-harris-corner-detector-32a88850b3f6)

2.1 Harris Corner Detection Method

Harris Corner detection algorithm was developed to identify the internal corners of an image. The corners of an image are basically identified as the regions in which there are variations in large intensity of the gradient in all possible dimensions and directions. Corners extracted can be a part of the image features, which can be matched with features of other images, and can be used to extract accurate information. Harris Corner Detection is a method to extract the corners from the input image and to extract features from the input image.

About the Function Used:

Syntax: cv2.cornerHarris(src, dest, blockSize, kSize, freeParameter, borderType)

Parameters:

- src – Input Image (Single-channel, 8-bit or floating-point)
- dest – Image to store the Harris detector responses. Size is same as source image
- blockSize – Neighborhood size (for each pixel value blockSize * blockSize neighbourhood is considered)
- ksize – Aperture parameter for the Sobel() operator
- freeParameter – Harris detector free parameter
- borderType – Pixel extrapolation method (the extrapolation mode used returns the coordinate of the pixel corresponding to the specified extrapolated pixel)

Demonstration

In [8]:
```
image = cv2.imread('color.jpg')
image = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
plt.figure(figsize=(15,10))
plt.title('Original Image')
plt.imshow(image)
plt.axis('off')
plt.show()
```

In [9]:
```
# convert the input image into grayscale color space
operatedImage = cv2.cvtColor(image,cv2.COLOR_RGB2GRAY)
plt.figure(figsize=(15,10))
plt.title('Grayscale Image')
plt.imshow(operatedImage,cmap='gray')
plt.axis('off')
plt.show()
```

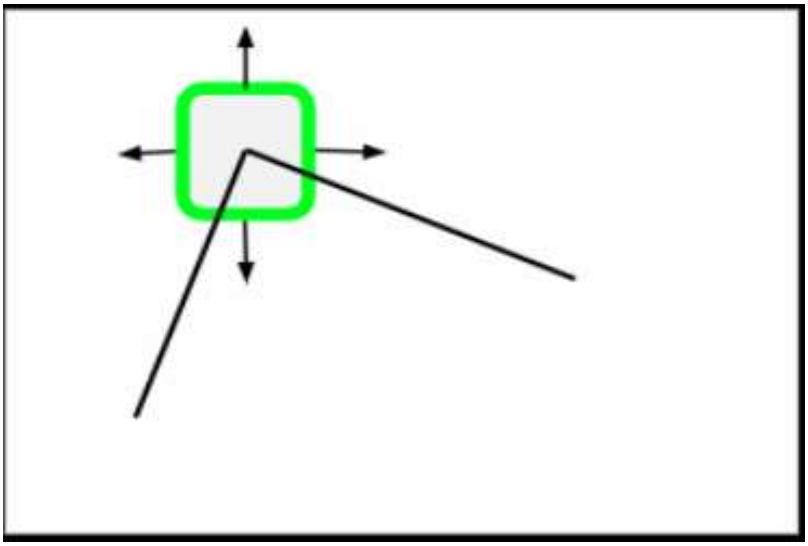
In [10]:
```
# modify the data type setting to 32-bit floating point
operatedImage = np.float32(operatedImage)
# apply the cv2.cornerHarris method to detect the corners with appropriate values
dest = cv2.cornerHarris(operatedImage,2,5,0.07)
# Results are marked through the dilated corners
dest = cv2.dilate(dest,None)
# Reverting back to the original image, with optimal threshold value
image[dest > 0.01 * dest.max()] = [0,251,255]
# Display the resultant image
plt.figure(figsize=(15,10))
plt.title('Image with Borders')
plt.imshow(image)
plt.axis('off')
plt.show()
```

2.2 Shi-Tomasi Corner Detection Method

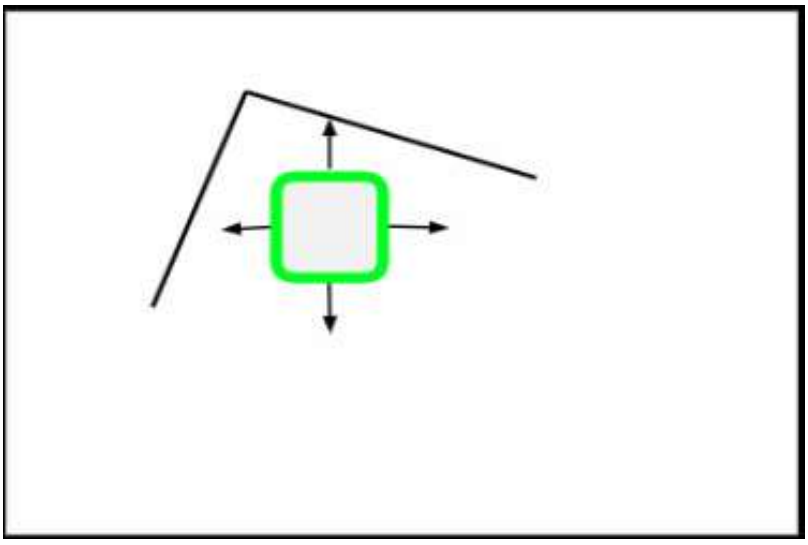
Shi-Tomasi Corner Detection was published by J.Shi and C.Tomasi in their paper 'Good Features to Track'. Here the basic intuition is that corners can be detected by looking for significant change in all direction.

We consider a small window on the image then scan the whole image, looking for corners.

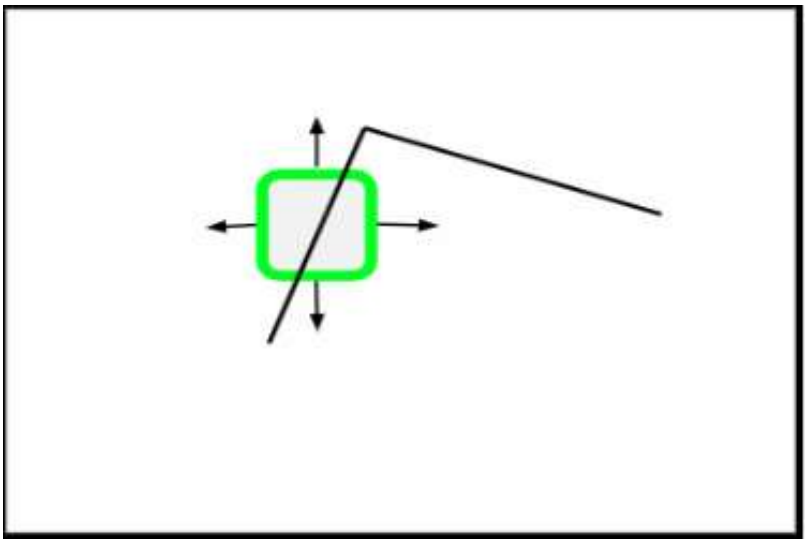
Shifting this small window in any direction would result in a large change in appearance, if that particular window happens to be located on a corner.



Flat regions will have no change in any direction.



If there's an edge, then there will be no major change along the edge direction.



Mathematical Overview:

For a window(W) located at (X, Y) with pixel intensity I(X, Y), formula for Shi-Tomasi Corner Detection is –

$$f(X,Y) = \Sigma(I(Xk,Yk) - I(Xk+\Delta X, Yk+\Delta Y))^2$$

where (Xk, Yk) ϵ W

According to the formula:

If we're scanning the image with a window just as we would with a kernel and we notice that there is an area where there's a major change no matter in what direction we actually scan, then we have a good intuition that there's probably a corner there.

Calculation of f(X, Y) will be really slow. Hence, we use Taylor expansion to simplify the scoring function, R.

$$R = min(\lambda 1, \lambda 2)$$

where λ1, λ2 are eigenvalues of resultant matrix.

Using goodFeaturesToTrack() Function:

Syntax: cv2.goodFeaturesToTrack(gray_img, maxc, Q, minD)

Parameters:

- gray_img – Grayscale image with integral values
- maxc – Maximum number of corners we want(give negative value to get all the corners)
- Q – Quality level parameter(preferred value=0.01)
- maxD – Maximum distance(preferred value=10)

Demonstration:

In [11]:
```
image = cv2.imread('color.jpg')
image = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
plt.figure(figsize=(15,10))
plt.title('Original Image')
plt.imshow(image)
plt.axis('off')
plt.show()
```

In [12]:
```
# convert the input image into grayscale color space
operatedImage = cv2.cvtColor(image,cv2.COLOR_RGB2GRAY)
plt.figure(figsize=(15,10))
plt.title('Grayscale Image')
plt.imshow(operatedImage,cmap='gray')
plt.axis('off')
plt.show()
```

In [13]:
```
# Shi-Tomasi corner detection function
# We are detecting only 100 best corners here
# You can change the number to get desired result.
corners = cv2.goodFeaturesToTrack(operatedImage,100,0.01,10)
# convert corners values to integer
# So that we will be able to draw circles on them
corners = np.int0(corners)
# draw lighr blue color circles on all corners
for i in corners:
    x, y = i.ravel()
    cv2.circle(image, (x, y), 3, (0, 251, 255), -1)
# Display the resultant image
plt.figure(figsize=(15,10))
plt.title('Image with Borders')
plt.imshow(image)
plt.axis('off')
plt.show()
```

3. Image Filtering, Edge Detection and Corner Detection in Videos

Write the following code for performing Image Filtering, Edge Detection and Corner Detection in videos:

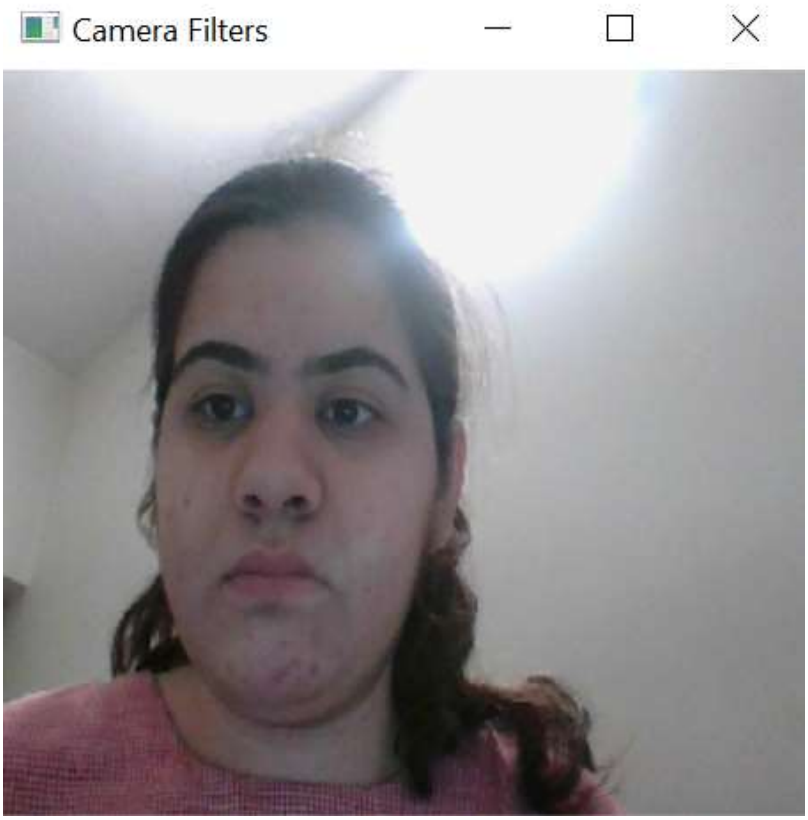
Code:

Filter_Edge_Corner.py

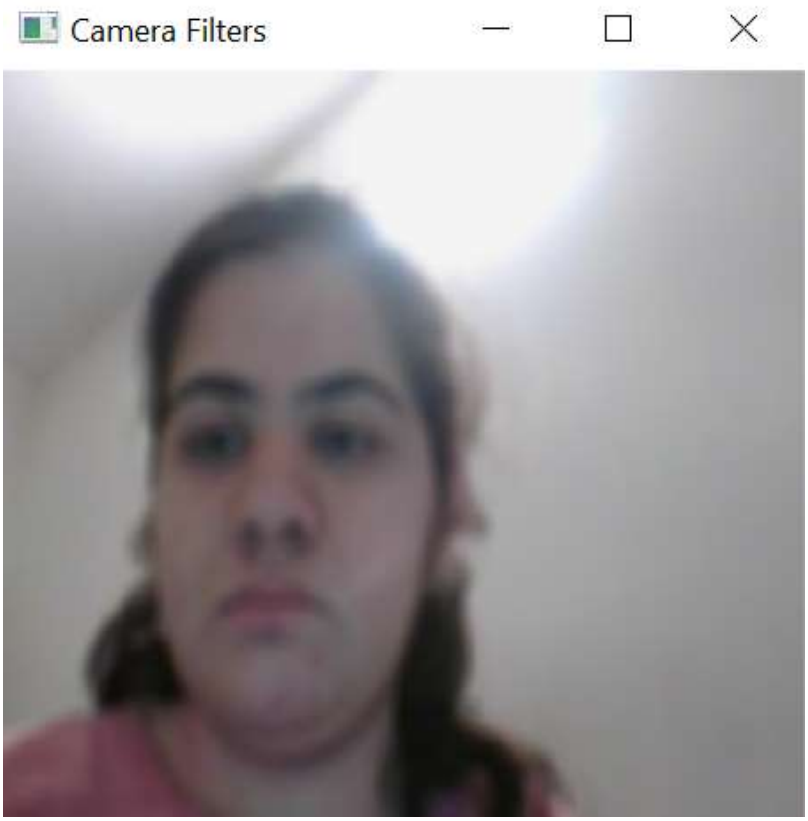
```python
import cv2
import numpy

PREVIEW  = 0   # Preview Mode
BLUR     = 1   # Blurring Filter
FEATURES = 2   # Corner Feature Detector
CANNY    = 3   # Canny Edge Detector

feature_params = dict( maxCorners = 500,
                       qualityLevel = 0.2,
                       minDistance = 15,
                       blockSize = 9)

image_filter = PREVIEW
alive = True

win_name = 'Camera Filters'
cv2.namedWindow(win_name, cv2.WINDOW_NORMAL)
result = None

source = cv2.VideoCapture(0)

while alive:
    has_frame, frame = source.read()
    if not has_frame:
        break

    frame = cv2.flip(frame,1)

    if image_filter == PREVIEW:
        result = frame
    elif image_filter == CANNY:
        result = cv2.Canny(frame, 80, 150)
    elif image_filter == BLUR:
        result = cv2.blur(frame, (13,13))
    elif image_filter == FEATURES:
        result = frame
        frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        corners = cv2.goodFeaturesToTrack(frame_gray, **feature_params) # list of corners detected in the image
        if corners is not None:
            corners = numpy.int0(corners)
            # draw red color circles on all corners
            for i in corners:
                x, y = i.ravel()
                cv2.circle(result, (x, y), 3, (0, 251, 255), -1)

    cv2.imshow(win_name, result)

    key = cv2.waitKey(1)
    if key == ord('Q') or key == ord('q') or key == 27:
        alive = False
    elif key == ord('C') or key == ord('c'):
        image_filter = CANNY
    elif key == ord('B') or key == ord('b'):
        image_filter = BLUR
    elif key == ord('F') or key == ord('f'):
        image_filter = FEATURES
    elif key == ord('P') or key == ord('p'):
        image_filter = PREVIEW

source.release()
cv2.destroyWindow(win_name)
```

Output:

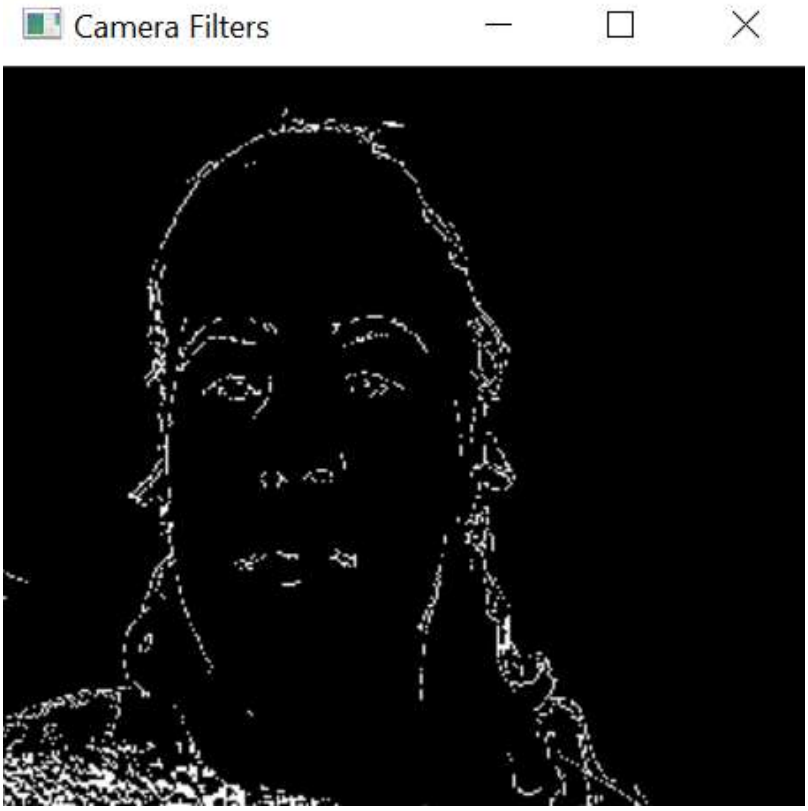
1. Preview:



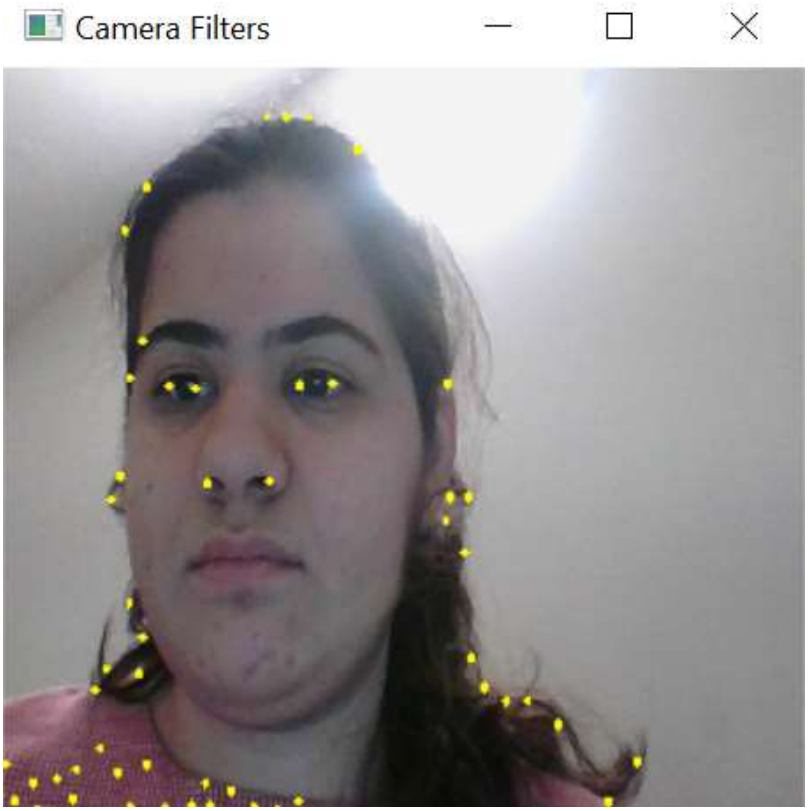
2. Blur:



3. Edge Detection:



4. Corner Detection:



The End of Chapter 4 !!!