Machine Vision Expriment 1

September 15, 2025

[ ]:

```python
#GREEN & INTELLIGENT TRANSPORTATION 2401 / GUO JIATONG
#Jupyter does not support any form of Chinese characters appearing in its exported PDF files.
#To prevent garbled text in PDF documents, all code comments are written in Chinese and translated into English.
```

[8]:

```python
#Input & Show Images
import cv2
import matplotlib.pyplot as plt

#Read Images
img_path1 = '/Users/guo2006/myenv/Machine Vision Experiment/bxc.jpg'
img_path2 = '/Users/guo2006/myenv/Machine Vision Experiment/background.jpg'
img_bxc = cv2.imread(img_path1) #Default: OpenCV Read Images in BGR Order
img_background = cv2.imread(img_path2)
if img_bxc is None or img_background is None:
    raise FileNotFoundError(f'Image Not Found!Check the path to retry.')

#BGR--->RGB,use matplotlib to show images
img_bxc = cv2.cvtColor(img_bxc,cv2.COLOR_BGR2RGB)
img_background = cv2.cvtColor(img_background,cv2.COLOR_BGR2RGB)

#Show
fig, ax = plt.subplots(1, 2, figsize=(10, 4))
ax[0].imshow(img_bxc)
ax[0].set_title('BXC Original')
ax[0].axis('on')

ax[1].imshow(img_background)
ax[1].set_title('Background')
ax[1].axis('off')

plt.tight_layout()
plt.show()
#You can use OpenCV's own GUI to show images.
#cv2.imshow('BXC Original Picture',img_bxc)
#cv2.imshow('Background',img_background)
```

```
#cv2.waitKey(0)
#cv2.destroyAllWindows
```

[2]:

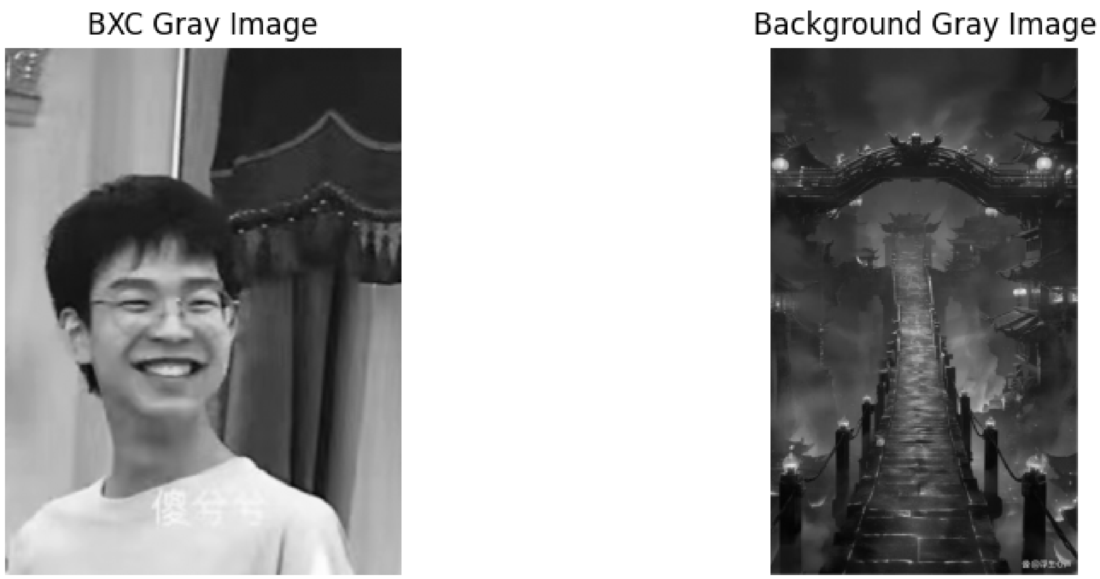
```
#Output the Size of images (Height,Width,Channels)
import cv2
import matplotlib.pyplot as plt

#Read the Images
img_path1 = '/Users/guo2006/myenv/Machine Vision Experiment/bxc.jpg'
img_path2 = '/Users/guo2006/myenv/Machine Vision Experiment/background.jpg'
img_bxc = cv2.imread(img_path1)
img_background = cv2.imread(img_path2)
if img_bxc is None or img_background is None:
    raise FileNotFoundError(f'Image Not Found!Check the path to retry.')

img_bxc = cv2.cvtColor(img_bxc,cv2.COLOR_BGR2RGB)
#img_background = cv2.cvtColor(img_background,cv2.BGR2RGB)

height_1, width_1, channels_1 = img_bxc.shape
height_2, width_2, channels_2 = img_background.shape

print(f'Image1 Size: {height_1} * {width_1} * {channels_1}')
print(f'Image1-> Height: {height_1},Width: {width_1},Channels: {channels_1}')
print(f'Image2 Size: {height_2} * {width_2} * {channels_2}')
print(f'Image2-> Height: {height_2},Width: {width_2},Channels: {channels_2}')
```

```
Image1 Size: 1165 * 874 * 3
```

```
Image1-> Height: 1165,Width: 874,Channels: 3
Image2 Size: 1001 * 584 * 3
Image2-> Height: 1001,Width: 584,Channels: 3
```

[3]:

```python
#Gray Images Management
import cv2
import matplotlib.pyplot as plt
#Input Images
img_path1 = '/Users/guo2006/myenv/Machine Vision Experiment/bxc.jpg'
img_path2 = '/Users/guo2006/myenv/Machine Vision Experiment/background.jpg'
img_bxc = cv2.imread(img_path1)
img_background = cv2.imread(img_path2)
if img_bxc is None or img_background is None:
    raise FileNotFoundError(f'Image Not Found!Check the path to retry.')
else:
    img_bxc_gray = cv2.cvtColor(img_bxc,cv2.COLOR_BGR2GRAY)
    img_background_gray = cv2.cvtColor(img_background,cv2.COLOR_BGR2GRAY)

    fig,ax = plt.subplots(1,2,figsize=(10,4))
    ax[0].imshow(img_bxc_gray,cmap='gray')
    ax[0].set_title('BXC Gray Image')
    ax[0].axis('off')

    ax[1].imshow(img_background_gray,cmap='gray')
    ax[1].set_title('Background Gray Image')
    ax[1].axis('off')
```

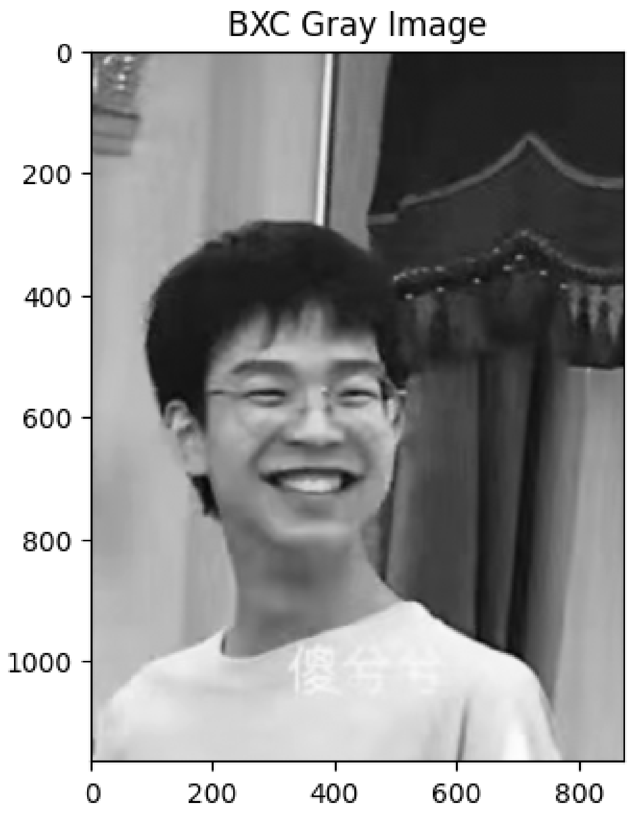


BXC Gray Image

Background Gray Image

[14]:

```python
#Gray Images Transform(Use plt to Output)
import cv2
import matplotlib.pyplot as plt
#Input Images
img_path = '/Users/guo2006/myenv/Machine Vision Experiment/bxc.jpg'
img_bxc = cv2.imread(img_path)
if img_bxc is None:
    raise FileNotFoundError(f'Image Not Found!Check the path to retry.')
else:
    img_bxc_gray = cv2.cvtColor(img_bxc,cv2.COLOR_BGR2GRAY)
    plt.imshow(img_bxc_gray,cmap='gray')
    plt.axis('on')
    plt.title('BXC Gray Image')
    plt.show()
```

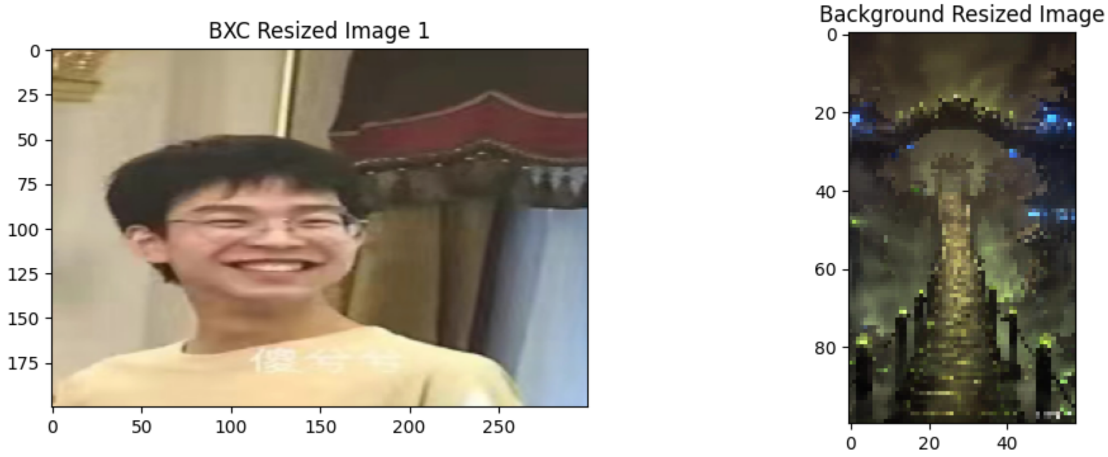


[4]:

```python
#Images Zoom operation
import cv2
import matplotlib.pyplot as plt

img_path1 = '/Users/guo2006/myenv/Machine Vision Experiment/bxc.jpg'
img_path2 = '/Users/guo2006/myenv/Machine Vision Experiment/background.jpg'
```

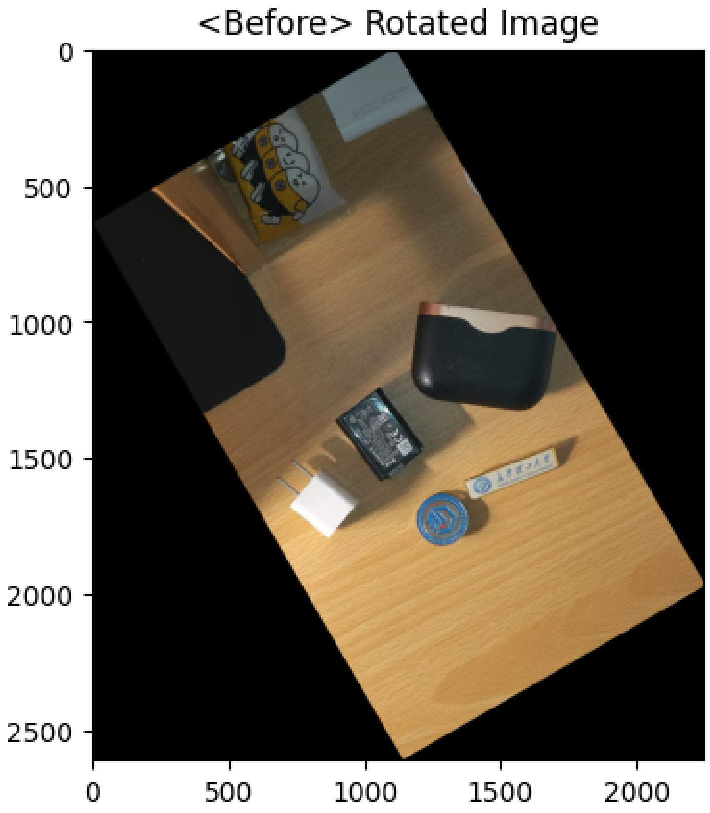
```
img_bxc = cv2.imread(img_path1)
img_background = cv2.imread(img_path2)

img_bxc = cv2.cvtColor(img_bxc, cv2.COLOR_BGR2RGB)

img_bxc_resized = cv2.resize(img_bxc,(300,200)) #Specify Size
#------------------------#
scale = 0.1 #Specify Zoom Ratio
img_background_resized = cv2.resize(img_background,None,fx = scale,fy = scale)

fig,ax = plt.subplots(1,2,figsize=(12,4))
ax[0].imshow(img_bxc_resized)
ax[0].set_title('BXC Resized Image 1')
ax[1].imshow(img_background_resized)
ax[1].set_title('Background Resized Image')
```

```
[4]: Text(0.5, 1.0, 'Background Resized Image')
```



```
[5]: #Rotation operation(Difficult)
import cv2
import matplotlib.pyplot as plt

img_path = '/Users/guo2006/myenv/Machine Vision Experiment/before.jpg'
img_before = cv2.imread(img_path)
if img_before is None:
    raise FileNotFoundError(f'Image Not Found!Check the path to retry.')
else:
    img_before = cv2.cvtColor(img_before, cv2.COLOR_BGR2RGB)
    height, width = img_before.shape[:2] #Slice the three digit array returned by .shape,discarding channel values
```

```
center = (width//2, height//2) #Find the Rotation Center
rotation_angle = 30 #Rotate counterclockwise 30deg
scale = 1 #Zoom Ratio

#Calculate the rotation matrix
M = cv2.getRotationMatrix2D(center, rotation_angle, scale)
#img_before_rotated = cv2.warpAffine(img_before, M, (width,height))␣
↪#Updated Matrix Resampling

#cv2.imshow('<before> Rotated Picture',img_before_rotated)
#cv2.waitKey(0)
#cv2.destroyAllWindows() #The four corners displayed will be cropped due to␣
↪insufficient canvas size.

#Automatically resize the canvas.
cos, sin = abs(M[0,0]), abs(M[0,1])
new_width = int(height*sin + width*cos)
new_height = int(width*sin + height*cos)
#Compensate for the offset
M[0,2] += new_width/2 - center[0] #center[0]--> width//2
M[1,2] += new_height/2 -center[1] #center[1]--> height//2
#Resample using the updated matrix M and the expanded canvas size␣
↪(new_width, new_height).
img_before_rotated = cv2.warpAffine(img_before, M ,(new_width,new_height))

plt.imshow(img_before_rotated)
plt.title('<Before> Rotated Image')
plt.axis('on')
plt.show()
```



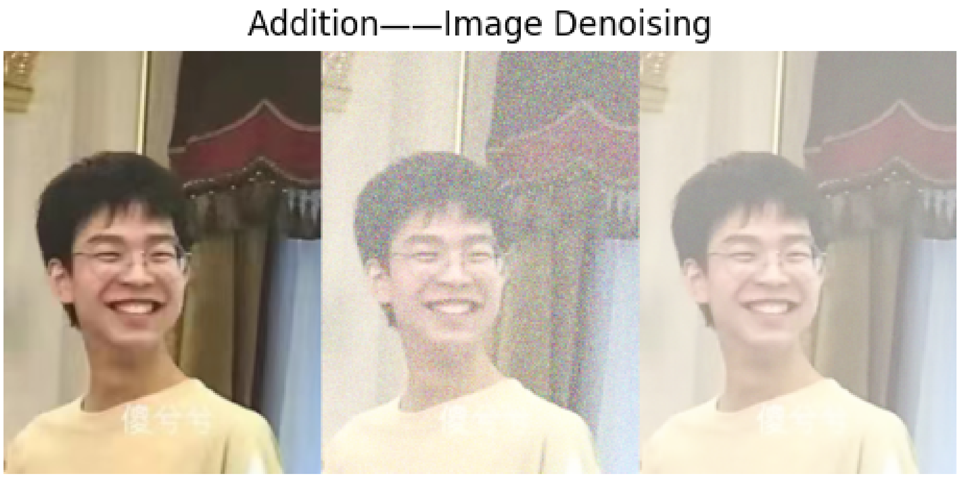
[19]:

```
#Cutting operation
import cv2
import matplotlib.pyplot as plt

img_path = '/Users/guo2006/myenv/Machine Vision Experiment/after.jpg'
img_after = cv2.imread(img_path)
if img_after is None:
    raise FileNotFoundError(f'Image Not Found!Check the path to retry.')
else:
    img_after = cv2.cvtColor(img_after, cv2.COLOR_BGR2RGB)
    y1, y2 = 50, 820 #Row's range
    x1, x2 = 115, 1530 #Column's range

    img_after_cropped = img_after[y1:y2, x1:x2]

    plt.imshow(img_after_cropped)
    plt.axis('off')
    plt.title('<After> Cropped Image')
    plt.show()
```

[6]:

```python
#Image Addition Operation -> Average Noise Reduction
import cv2
import matplotlib.pyplot as plt
import numpy as np
import os
import random

#Show Function(more convenient way to display multiple image comparisons)
def show(title, *imgs):
    #Multiple images are horizontally arranged in a row and displayed at the same height.
    h0 = imgs[0].shape[0] #The first image imported is at a uniform height.

    resized_list = []
    for img in imgs:
        resized_img = cv2.resize(img,(int(img.shape[1]*h0/img.shape[0]),h0))
        resized_list.append(resized_img)
    canvas = cv2.hconcat(resized_list)
    canvas = cv2.cvtColor(canvas, cv2.COLOR_BGR2RGB)

    plt.imshow(canvas)
    plt.title(title)
    plt.axis('off')
    plt.show()
```

```
#Image Averaging Function (Noise Reduction)
def add_noise(img,sigma=25):
    #Add Gauss Noise
    noise = np.random.normal(0,sigma,img.shape).astype(np.uint8)
    return cv2.add(img, noise)

img_path = '/Users/guo2006/myenv/Machine Vision Experiment/bxc.jpg'
img_bxc_original = cv2.imread(img_path)
height_bxc , width_bxc = img_bxc_original.shape[:2]
N = 50 #Generate 50 noise images

noise_img_list = [] #Generate a set of random noise list
for _ in range(N):
    noise_img_list.append(add_noise(img_bxc_original))
#Generate a set of random noise images
avg_noised = np.mean(noise_img_list, axis=0).astype(np.uint8)

show('Addition--Image Denoising',img_bxc_original,noise_img_list[0],avg_noised)
```

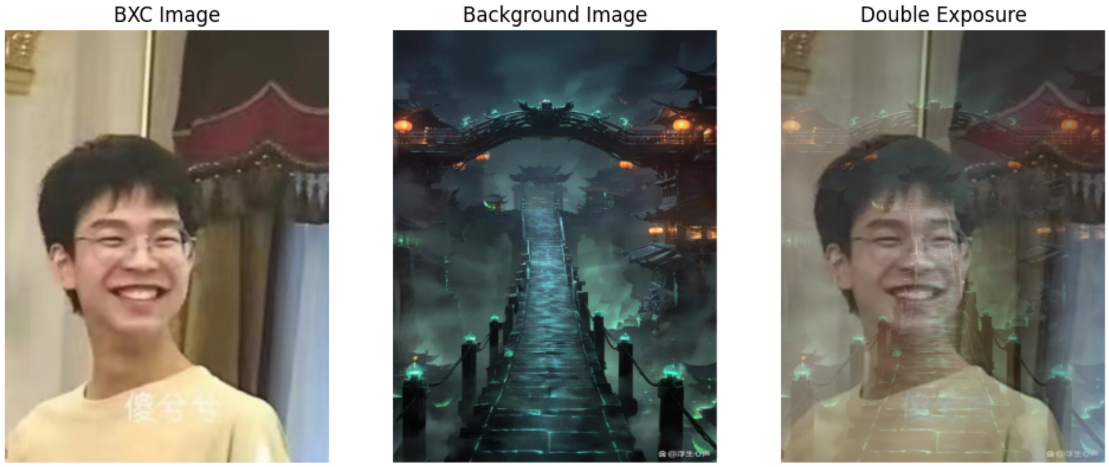


[11]:
```
#Image Addition Operation -> Double Exposure
import cv2
import matplotlib.pyplot as plt
import numpy as np
import os
import random

img_bxc_path = '/Users/guo2006/myenv/Machine Vision Experiment/bxc.jpg'
img_background_path = '/Users/guo2006/myenv/Machine Vision Experiment/
 ↪background.jpg'
img_bxc = cv2.imread(img_bxc_path)
```

```python
img_background = cv2.imread(img_background_path)
#Color Mode Conversion
img_bxc = cv2.cvtColor(img_bxc, cv2.COLOR_BGR2RGB)
img_background = cv2.cvtColor(img_background, cv2.COLOR_BGR2RGB)

height_bxc , width_bxc = img_bxc.shape[:2]
#Adjust the background image to match the size of the main image.
img_background = cv2.resize(img_background,(width_bxc,height_bxc))
#Double Exposure Weighting
double_exposure = cv2.addWeighted(img_bxc,0.6,img_background,0.4,0)

fig, ax = plt.subplots(1,3,figsize=(12,5))
ax[0].imshow(img_bxc)
ax[0].set_title('BXC Image')
ax[0].axis('off')
ax[1].imshow(img_background)
ax[1].set_title('Background Image')
ax[1].axis('off')
ax[2].imshow(double_exposure)
ax[2].set_title('Double Exposure')
ax[2].axis('off')
```

[11]:

```
(np.float64(-0.5), np.float64(873.5), np.float64(1164.5), np.float64(-0.5))
```



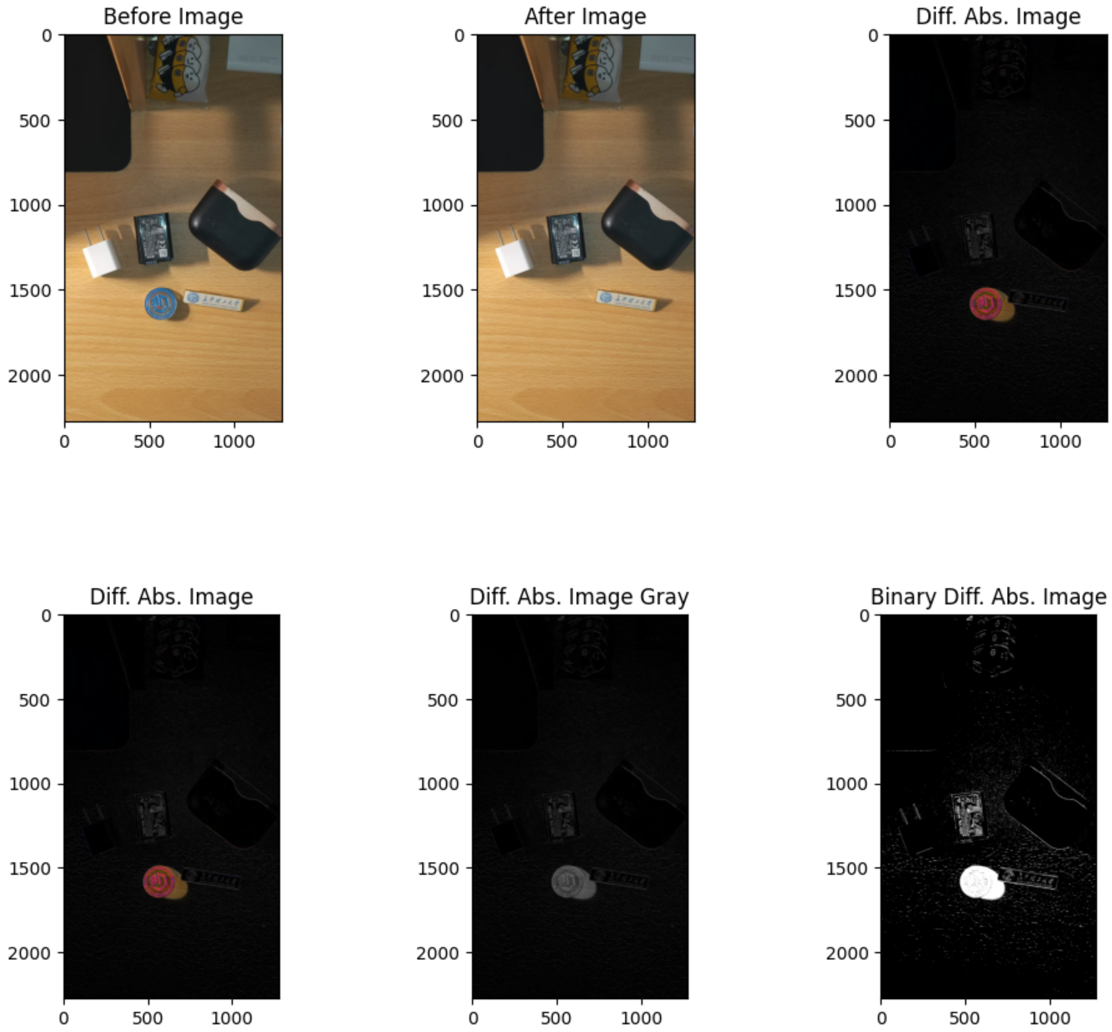
[6]:

```python
#Subtraction -> Still Life Difference Recognition
import cv2
import matplotlib.pyplot as plt
import numpy as np
import os
import random
```

```python
img_path_before = '/Users/guo2006/myenv/Machine Vision Experiment/before.jpg'
img_path_after = '/Users/guo2006/myenv/Machine Vision Experiment/after.jpg'
img_before, img_after = cv2.imread(img_path_before), cv2.imread(img_path_after)

#Modify to a uniform size
height_before, width_before = img_before.shape[:2]
img_after = cv2.resize(img_after,(width_before, height_before))
#Create a difference plot
img_diff_abs = cv2.absdiff(img_before, img_after)
img_diff_abs_gray = cv2.cvtColor(img_diff_abs, cv2.COLOR_BGR2GRAY)
#Use the Otsu algorithm to calculate the optimal threshold.
thresh_used, mask = cv2.threshold(img_diff_abs_gray,0,255,cv2.THRESH_BINARY +␣
 ↪cv2.THRESH_OTSU)
print('Otsu Automatic Threshold=',thresh_used)

#Color Mode Conversion
img_before = cv2.cvtColor(img_before, cv2.COLOR_BGR2RGB)
img_after = cv2.cvtColor(img_after, cv2.COLOR_BGR2RGB)
img_diff_abs = cv2.cvtColor(img_diff_abs, cv2.COLOR_BGR2RGB)
img_diff_abs_gray = cv2.cvtColor(img_diff_abs_gray, cv2.COLOR_BGR2RGB)
mask = cv2.cvtColor(mask, cv2.COLOR_BGR2RGB)

fig, ax = plt.subplots(1,3,figsize=(12,4))
ax[0].imshow(img_before)
ax[0].set_title('Before Image')
ax[1].imshow(img_after)
ax[1].set_title('After Image')
ax[2].imshow(img_diff_abs)
ax[2].set_title('Diff. Abs. Image')
fig, ax = plt.subplots(1,3,figsize=(12,4))
ax[0].imshow(img_diff_abs)
ax[0].set_title('Diff. Abs. Image')
ax[1].imshow(img_diff_abs_gray)
ax[1].set_title('Diff. Abs. Image Gray')
ax[2].imshow(mask)
ax[2].set_title('Binary Diff. Abs. Image')
```

```
Otsu Automatic Threshold= 31.0
```

```
[6]: Text(0.5, 1.0, 'Binary Diff. Abs. Image')
```

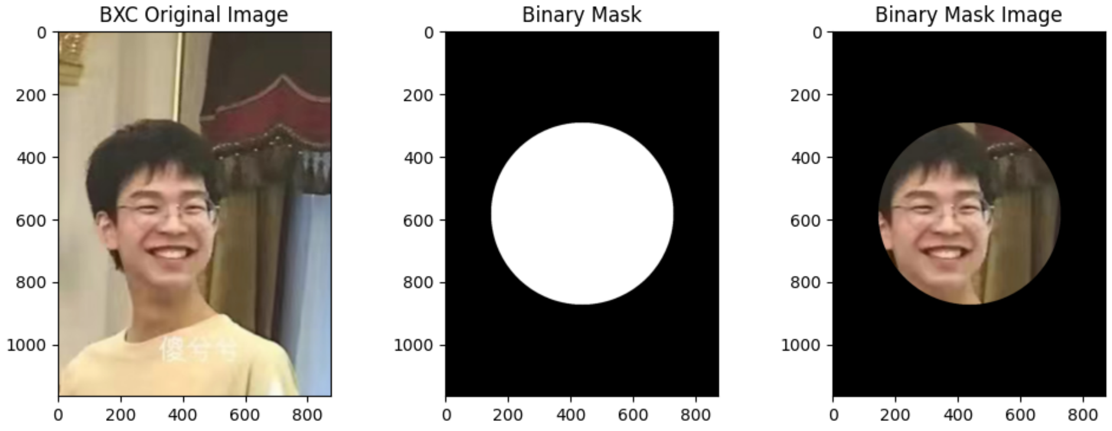


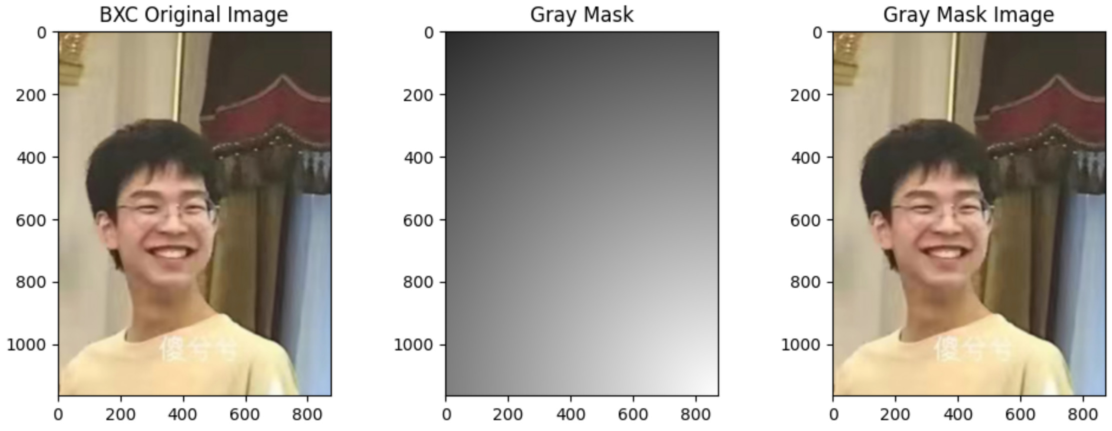
[30]:

```python
#Multiplication (Masked Clipping)
import cv2
import matplotlib.pyplot as plt
import random
import numpy as np
import os

img_bxc_path = '/Users/guo2006/myenv/Machine Vision Experiment/bxc.jpg'
img_bxc = cv2.imread(img_bxc_path)
height_bxc, width_bxc = img_bxc.shape[:2]
#Binary Mask
mask_bin = np.zeros((height_bxc,width_bxc),np.uint8) #Create a black mask
cv2.circle(mask_bin,(width_bxc//2,height_bxc//2),min(height_bxc, width_bxc)//3, 255, -1) #Make a white circular area appears in the center of the black mask.
```

```python
result_bin = cv2.bitwise_and(img_bxc, img_bxc, mask=mask_bin) #The white circular area within the mask is retained; all other parts are removed.

#Gray Mask
Y, X = np.ogrid[:height_bxc, :width_bxc] #Create an Euclidean coordinate grid
dist = np.sqrt((X-width_bxc)**2+(Y-height_bxc)**2) #Calculate the Euclidean distance between each pixel and the center.
#Map distances linearly to grayscale values between 0 and 255 (set mask rules)
mask_gray = np.clip((255-dist*(255/min(height_bxc, width_bxc)/2)),0,255).astype(np.uint8)
#Could this cause subtle changes in grayscale? How can this be resolved???
result_gray = cv2.bitwise_and(img_bxc, img_bxc, mask=mask_gray)

img_bxc = cv2.cvtColor(img_bxc, cv2.COLOR_BGR2RGB)
mask_bin = cv2.cvtColor(mask_bin, cv2.COLOR_BGR2RGB)
mask_gray = cv2.cvtColor(mask_gray, cv2.COLOR_BGR2RGB)
result_bin = cv2.cvtColor(result_bin, cv2.COLOR_BGR2RGB)
result_gray = cv2.cvtColor(result_gray, cv2.COLOR_BGR2RGB)

fig, ax = plt.subplots(1,3,figsize=(12,4))
ax[0].imshow(img_bxc)
ax[0].set_title('BXC Original Image')
ax[1].imshow(mask_bin)
ax[1].set_title('Binary Mask')
ax[2].imshow(result_bin)
ax[2].set_title('Binary Mask Image')
fig, ax = plt.subplots(1,3,figsize=(12,4))
ax[0].imshow(img_bxc)
ax[0].set_title('BXC Original Image')
ax[1].imshow(mask_gray)
ax[1].set_title('Gray Mask')
ax[2].imshow(result_gray)
ax[2].set_title('Gray Mask Image')
```

```
[30]: Text(0.5, 1.0, 'Gray Mask Image')
```

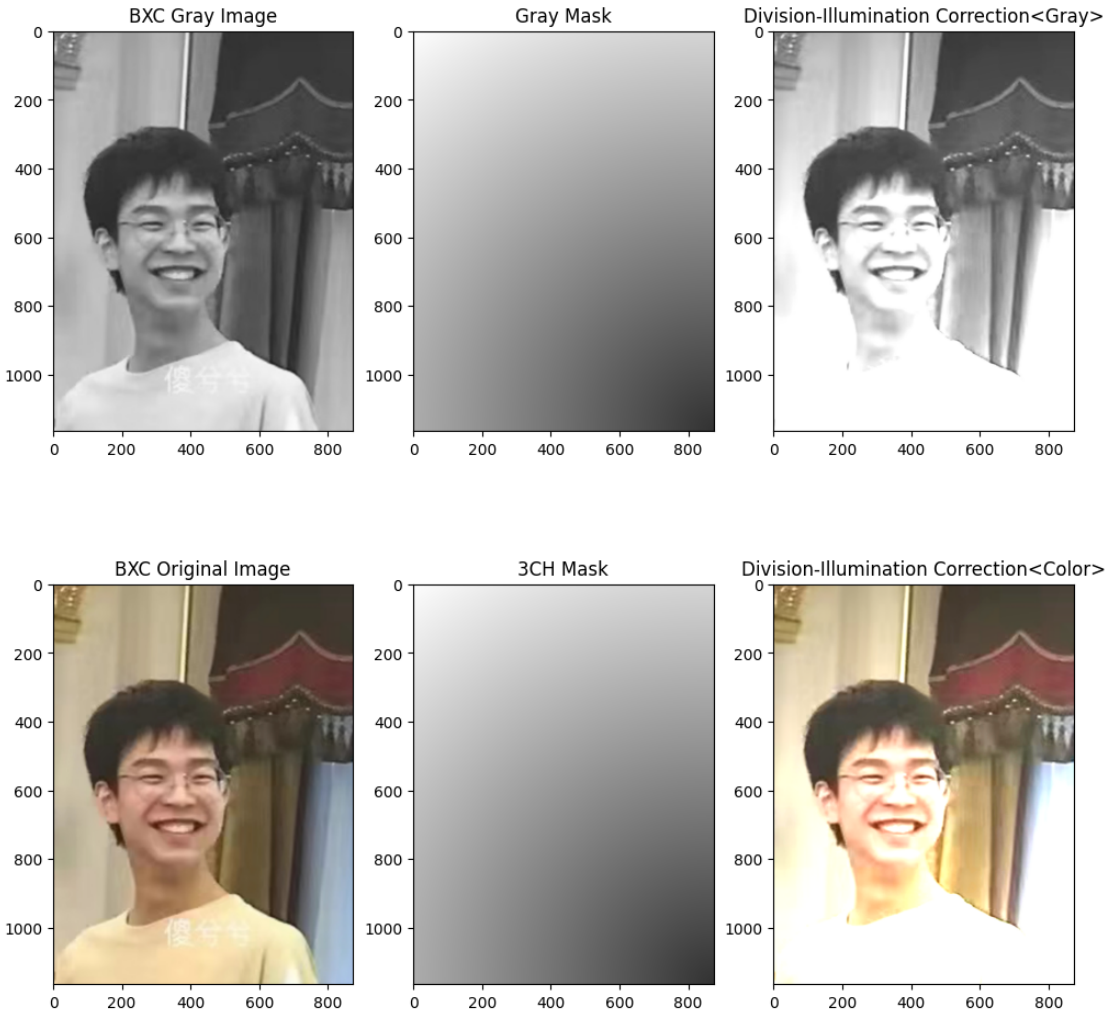




[42]:
```python
#Division
import cv2
import matplotlib.pyplot as plt
import random
import os
import numpy as np

#Constant Division--->Darken the entire image uniformly
img_bxc_path = '/Users/guo2006/myenv/Machine Vision Experiment/bxc.jpg'
img_bxc = cv2.imread(img_bxc_path).astype(np.uint8) #Convert to a floating-point number between 0 and 255 for convenient division.

img_bxc_dark = (img_bxc/3).clip(0,255).astype(np.uint8) #Integer division + truncation + conversion
img_bxc_dark = cv2.cvtColor(img_bxc_dark, cv2.COLOR_BGR2RGB)
```

```python
#Non-integer division method--->Compensating for uneven illumination (image correction)
height_bxc, width_bxc = img_bxc.shape[:2]
Y, X = np.ogrid[:height_bxc, :width_bxc] #Create grid

light_illum = np.sqrt((X-width_bxc)**2+(Y-height_bxc)**2) #Calculate the Euclidean distance between each pixel and the center.
light_illum = (light_illum/light_illum.max()*0.8+0.2).astype(np.float32) #Normalized to 0.2-1.0
#light_illum-->2CH, img_bxc-->3CH, cannot be divied by using np, 2WAYS to solve this problem
#1.img_bxc--->img_bxc_gray(BGR->GRAY),Two 2CH images can be subtracted from each other. <2CH GRAY IMAGE>
#2.Transfer light_illum into 3CH. <3CH COLOR IMAGE>

#WAY1. 2CH GRAY IMAGE
img_bxc_gray = cv2.cvtColor(img_bxc, cv2.COLOR_BGR2GRAY)
#Subtract to compensate for dark areas + 0-255 clipping + format conversion
corrected_img_2ch = (img_bxc_gray / light_illum).clip(0,255).astype(np.uint8)

#WAY2. 3CH COLOR IMAGE
#Expand the light template to three channels
light_illum_3ch = cv2.merge([light_illum, light_illum, light_illum])
#Subtract to compensate for dark areas + 0-255 clipping + format conversion
corrected_img_3ch = (img_bxc / light_illum_3ch).clip(0,255).astype(np.uint8)

img_bxc = cv2.cvtColor(img_bxc, cv2.COLOR_BGR2RGB)
light_illum = cv2.cvtColor(light_illum, cv2.COLOR_BGR2RGB)
corrected_img_2ch = cv2.cvtColor(corrected_img_2ch, cv2.COLOR_BGR2RGB)
img_bxc_gray = cv2.cvtColor(img_bxc_gray, cv2.COLOR_BGR2RGB)
light_illum_3ch = cv2.cvtColor(light_illum_3ch, cv2.COLOR_BGR2RGB)
corrected_img_3ch = cv2.cvtColor(corrected_img_3ch, cv2.COLOR_BGR2RGB)

fig, ax = plt.subplots(2,3,figsize=(12,12))
ax[0,0].imshow(img_bxc_gray)
ax[0,0].set_title('BXC Gray Image')
ax[0,1].imshow((light_illum*255).astype(np.uint8))
ax[0,1].set_title('Gray Mask')
ax[0,2].imshow(corrected_img_2ch)
ax[0,2].set_title('Division-Illumination Correction<Gray>')
ax[1,0].imshow(img_bxc)
ax[1,0].set_title('BXC Original Image')
ax[1,1].imshow((light_illum_3ch*255).astype(np.uint8))
ax[1,1].set_title('3CH Mask')
ax[1,2].imshow(corrected_img_3ch)
ax[1,2].set_title('Division-Illumination Correction<Color>')
```

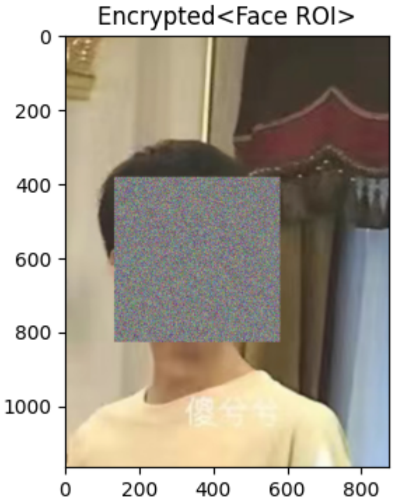
```
[42]: Text(0.5, 1.0, 'Division-Illumination Correction<Color>')
```

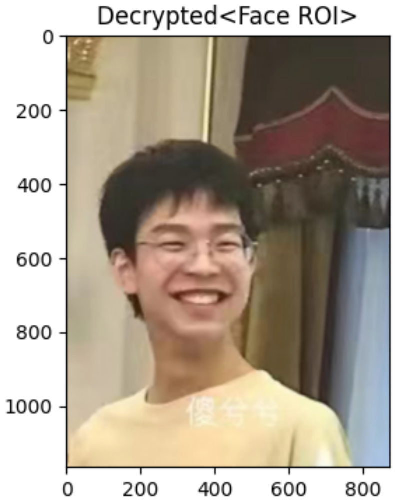


```
[7]: #ROI Decode
import cv2
import matplotlib.pyplot as plt
import random
import os
import numpy as np

#Load Face Detector
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')

img_face_path = '/Users/guo2006/myenv/Machine Vision Experiment/bxc.jpg'
face_img = cv2.imread(img_face_path)
```

```python
if face_img is None:
    raise FileNotFoundError(f'Image Not Found!Check the path to retry.')
face_img_gray = cv2.cvtColor(face_img,cv2.COLOR_BGR2GRAY)
#Detect face -> Obtain ROI mask
faces = face_cascade.detectMultiScale(face_img_gray, scaleFactor=1.2, minNeighbors=5)
#The return value `faces` is an N×4 numpy array, where each row (x, y, w, h) represents the top-left coordinates and dimensions of a face.

#Create a whole-black mask
mask = np.zeros(face_img.shape[:2], dtype=np.uint8)   # Single-channel full black
#Face whitening
for (x, y, w, h) in faces:
    cv2.rectangle(mask, (x, y), (x+w, y+h), 255, -1)
    #Draw on the mask, top-left corner, bottom-right corner, 255=white, solid fill
#Generate a random key
random_key = np.random.randint(0,256,face_img.shape,dtype=np.uint8) #0-255Random
#A 3D array with dimensions identical to face_img
#Perform XOR encryption only on the facial region (mask > 0 area).
encrypted = face_img.copy() #1.Copy the Original Image
encrypted[mask > 0] = cv2.bitwise_xor(face_img, random_key)[mask > 0] #2.Modify the facial features while the background unchanged.

#Perform XOR encryption again to decrypt.
decrypted = encrypted.copy()
decrypted[mask > 0] = cv2.bitwise_xor(encrypted, random_key)[mask > 0]

#Color Transfer
encrypted = cv2.cvtColor(encrypted, cv2.COLOR_BGR2RGB)
decrypted = cv2.cvtColor(decrypted, cv2.COLOR_BGR2RGB)

fig,ax = plt.subplots(1,2,figsize=(12,4))
ax[0].imshow(encrypted)
ax[0].set_title('Encrypted<Face ROI>')
ax[1].imshow(decrypted)
ax[1].set_title('Decrypted<Face ROI>')
```

```
[7]: Text(0.5, 1.0, 'Decrypted<Face ROI>')
```


Encrypted<Face ROI>
0
200
400
600
800
1000
0
200
400
600
800
傻兮兮


Decrypted<Face ROI>
0
200
400
600
800
1000
0
200
400
600
800
傻兮兮