

# 机器视觉-实践作业-1

国嘉通<sup>\*</sup>

2025 年 9 月 15 日

## 摘要

机器视觉第一次实验课程主要包含两大部分的练习，一是实验环境的搭建，二是基础的图像操作。对于实验环境的基本搭建部分，从基于 ANACONDA 的 Windows 系统及基于 Homebrew 的 Mac 系统的基础环境的搭建两方面分别展开。对于基础图像操作部分，将分多个部分进行展开讲解。此外，本文中夹杂着个人对于图像变换、运算等内在原理的思考及疑问。希望本篇文章是一篇内容翔实、方便随时翻阅复习学习 OpenCV 的学习笔记。另外，本文使用 L<sup>A</sup>T<sub>E</sub>X 书写完成，在附录部分有作者整理的精简 L<sup>A</sup>T<sub>E</sub>X 简单上手使用技巧，供读者及时翻阅。文章完成较为仓促，有许多细节作者无法在文中一一落实到位，敬请谅解。

长沙理工大学

卓越工程师学院 绿色智慧交通 2401 班

国嘉通

---

<sup>\*</sup>Tel.+86-166 8130 6085, E-mail:guojiatom2006@outlook.com

# 目录

<b>1 Python 环境配置</b>	<b>4</b>
1.1 Windows 系统情况下的相关配置	4
1.2 Mac 系统情况下的相关配置	5
1.3 配置完成后的 pip list 截图 (部分)	6
<b>2 OpenCV 基础图像操作</b>	<b>7</b>
2.1 读取与显示图像	7
2.2 图片属性输出	9
2.3 图片灰度转换操作	10
2.4 图像变换操作	12
2.4.1 图像缩放操作	12
2.4.2 图像旋转操作	14
2.4.3 图像剪裁操作	16
2.5 图像运算操作	18
2.5.1 加法运算	18
2.5.2 减法运算	21
2.5.3 乘法运算	23
2.5.4 除法运算	25
2.6 逻辑运算	28
<b>3 L<sup>A</sup>T<sub>E</sub>X 使用基础命令</b>	<b>31</b>
3.1 代码基本结构	31
3.2 单双栏版式变换	32
3.3 公式插入	33
3.3.1 普通公式	33
3.3.2 矩阵	36
3.3.3 转义符号	37
3.4 枚举	37
3.5 图片插入	38
3.5.1 单图插入	38
3.5.2 多图插入	38
3.6 表格插入	39
3.7 文本框插入	40
3.7.1 基于 \fbox 简单文本框	40
3.7.2 基于 \tcolorbox 复杂文本框	41
3.8 引用插入	41

3.8.1	正文引用	41
3.8.2	文献引用	41

# 1 Python 环境配置

## 1.1 Windows 系统情况下的相关配置

使用Anaconda 作为 Python 内核，搭配 VSCode 作为编辑器可完成相关设置。具体步骤如下：

1. 在 Anaconda Prompt 中搭建环境，指令为“conda create -n 环境名 python=版本号”。  
例如，欲想搭建一个名为myenv 的 Python 版本为3.11.11 的环境，我们可以输入以下指令：

```
conda create -n myenv python=3.11.11
```

2. 搭建完虚拟环境后，要将其激活。

激活指令为“conda activate 环境名”。假设已经以“myenv”搭建好环境，我们可以输入以下指令以进入此虚拟环境。（若不进行此步骤，后续安装包的过程中则会直接安装在 base 基环境下）

```
conda activate myenv
```

3. 完成环境激活后，则可以安装相应的软件包。

一些较为常见的软件包如下表 1 所示：

Package	Package	Package	Package
absl-py	cachetools	certifi	charset-normalizer
cloudpickle	colorama	contourpy	cycler
et-xmlfile	filelock	fonttools	google-auth
google-auth-oauthlib	grpcio	gym	gym-notices
idna	Jinja2	kiwisolver	Markdown
MarkupSafe	<b>matplotlib</b>	mpmath	networkx
<b>numpy</b>	oauthlib	<b>opencv-python</b>	openpyxl
packaging	<b>pandas</b>	<b>pathlib</b>	<b>pillow</b>
<b>pip</b>	protobuf	pyasn1	pyasn1 _ modules
pyparsing	<b>jupyter notebook</b>	pytz	<b>PyYAML</b>
requests	requests-oauthlib	rsa	<b>scipy</b>
<b>seaborn</b>	setuptools	six	sympy
<b>tensorboard</b>	tensorboardX	<b>tk</b>	<b>torch</b>
torchaudio	torchvision	<b>tqdm</b>	tzdata
urllib3	<b>opencv-python</b>	wheel	xlrd

可以使用“pip install Package”指令安装相关软件包，其中 Package 为表1中的软件包或其之外的软件包。假设欲安装numpy 软件包，我们可以输入如下指令：

```
pip install numpy
```

### 注意！

上述指令可能安装最新版本的对应的软件包或者是最适配当前 Python 版本的软件包。如果想要安装指定版本的软件包，应当使用“pip install `Package`==`Version`”指令。以安装 2.2.6 版本的 numpy 包为例：

```
pip install numpy==2.2.6
```

4. 其余的部分有用指令见下表<sup>2</sup>。

指令	代码
删除环境中的包	pip uninstall <code>Package</code>
删除环境	conda remove -n <code>Env.name</code>
退出虚拟环境（回到 base）	detective
查看虚拟环境中的包库	pip list
查看 Env 列表（可以用于改环境名）	conda env list

## 1.2 Mac 系统情况下的相关配置

对于 Mac 系统，我们可以直接使用系统带有的终端进行配置，而不需要安装 Anaconda。不过我们需要在终端中安装 Homebrew，此插件源为 Github，有时内网可能不灵，解决方案是多试几次或者科学上网（可以使用 Infiniport）。

1. 首先可以使用指令来检测 Mac 是否自带 Python。指令如下：

```
python -version (对于 Mac 自带 Python 版本为 2.x)
python3 -version (对于大部分 Mac Python 版本为 3.x)
```

2. 若无预置 Python，则需要通过 Homebrew 包管理器来安装最新的 Python 版本。如果还没有安装 Homebrew，应先安装它。（源在 Github，或许需要科学上网 > > [infiniport.xyz](https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)）在终端中输入：

```
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

3. Homebrew 安装完成后，使用其安装 Python。指令为：

```
brew install python
```

4. 为了避免不同项目之间的冲突，应搭建虚拟环境，并在虚拟环境中搭建对应的包。此步骤大体包含创建虚拟环境、激活虚拟环境、在虚拟环境中安装包、退出虚拟环境四部分。首先，创建虚拟环境。以搭建环境名为“myenv”的 Python 环境为例，其代码如下所示。

```
python3 -m venv myenv
```

然后，激活虚拟环境，代码如下所示。

```
source myenv/bin/activate
```

在所创建的虚拟环境中安装对应的软件包，以安装 `numpy` 包为例，代码如下所示。

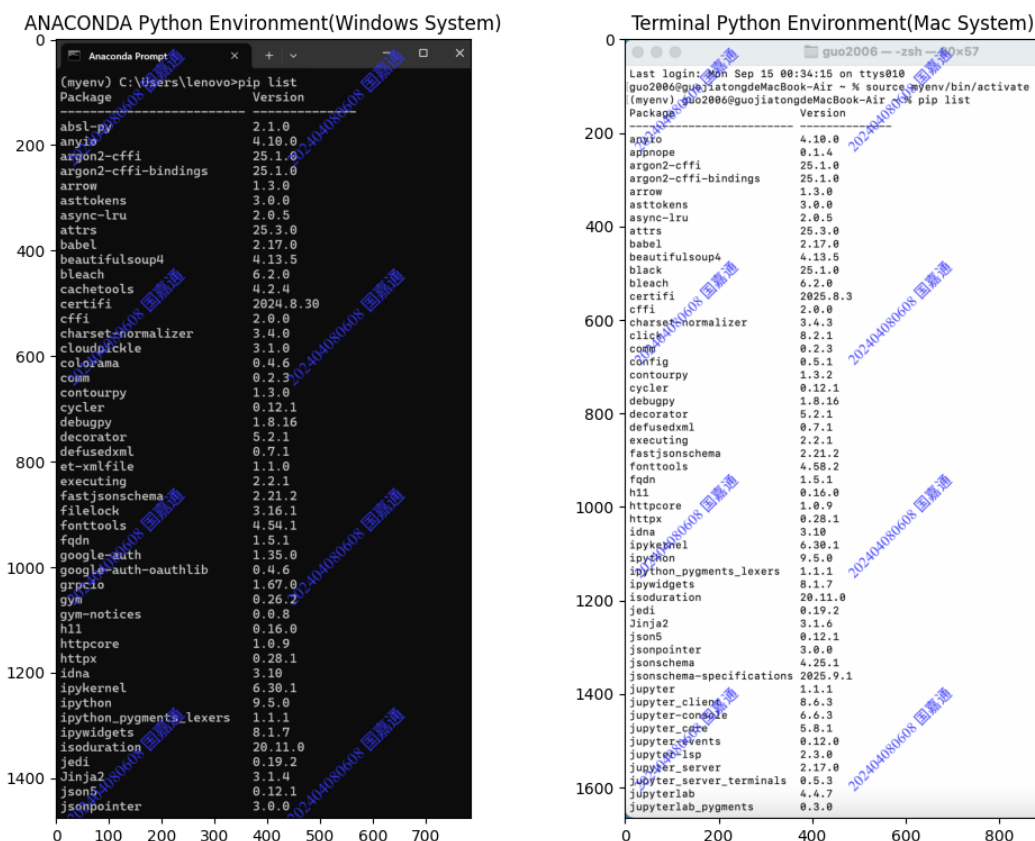
```
pip install numpy
```

最后，所有上述步骤完成后，可以使用 `deactivate` 指令退出虚拟环境，并在 IDE 中进行配置，不再过多赘述。

### 1.3 配置完成后的 `pip list` 截图 (部分)

最终，在 Windows 系统下的 Anaconda 中配置的 Python 环境所安装的软件包 (部分) 以及 Mac 系统下的终端-Homebrew 中配置的 Python 环境所安装的软件包 (部分) 如下图所示。

可以在激活对应环境后键入“`pip list`”以显示它。



## 2 OpenCV 基础图像操作

OpenCV 的基础操作包含读取 → 显示 → 属性输出 → 转换 → 变换 → 运算 → 解码等基本操作。本节利用 Jupyter Notebook 分别书写练习了相关语法。下面将一一展开进行叙述。

### 2.1 读取与显示图像

本小节主要展示如何使用 opencv-python 库以及 matplotlib 库中的 pyplot 来导入图像并显示图像。

```
# 导入图像并显示图像
import cv2
import matplotlib.pyplot as plt

# 读取图片
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) #opencv 默认以 BGR 顺序读入
img_background = cv2.imread(img_path2)
if img_bxc is None or img_background is None:
    raise FileNotFoundError(f'未读取到图片，请检查路径')

# 将 BGR 图像转换为 RGB 图像，使用 matplotlib 显示
img_bxc = cv2.cvtColor(img_bxc, cv2.COLOR_BGR2RGB)
img_background = cv2.cvtColor(img_background, cv2.COLOR_BGR2RGB)

# 显示
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()
# 可以使用 opencv 自带的 GUI
```

```
#cv2.imshow('BXC Original Picture',img_bxc)
#cv2.imshow('Background',img_background)
#cv2.waitKey(0)
#cv2.destroyAllWindows
```

- 导入图像——> 使用 `cv2.imread( 'PATH' )` 函数进行导入操作。  
\* 注：为方便后期的调用与修改，可以将 PATH 作为字符串存入新变量，然后再在 `cv2.imread(New Variables)` 中调用此新变量即可。

- 显示图像——> 有两种方式，如下所示：

1. 使用 **matplotlib** 库的 `pyplot` 绘制显示，值得注意的是，在 Jupyter Notebook 中，只支持此种 `plt` 形式的显示。若使用 `opencv-python` 库进行 `imshow` 会卡死报错。

其具体语法如下所示：（假设图片变量名为 `img`）

```
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
# 转换色彩格式很重要，cv2 读入为 BGR，而 plt 为 RGB.
plt.imshow(img) # 显示图片
plt.title('Image Title Name') # 图标题名
plt.axis('off/on') # 坐标轴有无
plt.show()
```

2. 使用 `opencv-python` 库的 `cv2` 绘制显示。此种方法无需进行 BGR—>RGB 转换，但不能在 Jupyter Notebook 中使用。

具体语法如下所示：（假设图片变量名为 `img`）

```
cv2.imshow('Image Title Name', img) # 图片名称及显示对象
cv2.waitKey(0) # 等待退出窗口的时间 (ms), 0 为无限等待
cv2.destroyAllWindows() # 按任意键退出所有窗口
```





## 2.2 图片属性输出

本小节主要展示如何输出给定图片的尺寸信息，包含其**高度** (Height)、**宽度** (Width) 以及**通道数** (Channels)。

```
# 输出图像尺寸（高度、宽度及通道数）
import cv2
import matplotlib.pyplot as plt

# 读取图像
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'未读取到图片，请重新检查路径')

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

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


```
print(f'图像 1 尺寸: {height_1} * {width_1} * {channels_1}')
print(f'图像 1 高度: {height_1}, 宽度:{width_1}, 通道数:{channels_1}')
print(f'图像 2 尺寸: {height_2} * {width_2} * {channels_2}')
print(f'图像 1 高度: {height_2}, 宽度:{width_2}, 通道数:{channels_2}')
```

图像 1 尺寸: 1165 \* 874 \* 3

图像 1 高度: 1165, 宽度:874, 通道数:3

图像 2 尺寸: 1001 \* 584 \* 3

图像 1 高度: 1001, 宽度:584, 通道数:3

本小节主要使用.shape 语法, 获得图片尺寸数据。

其中, 有:

$$ImageData = img.shape = \begin{bmatrix} Height & Width & Channels \end{bmatrix}$$

也就是说, 使用 img.shape 得到的是一个 1×3 的矩阵, 分别为图片的高度、宽度、通道数。

之后, 我们再使用 print() 函数进行打印输出即可。

后续当我们只需要使用图片的高度和宽度数据, 而不需要通道数数据时, 可以对上面矩阵进行切片。

例如:

```
img_height, img_width = img.shape[: 2] # 只切片使用前两个数据
```

## 2.3 图片灰度转换操作

灰度转换操作在图像预处理部分是十分重要且实用的。其内容较为简单, 本部分将详细阐述。

```
# 灰度图像处理
import cv2
import matplotlib.pyplot as plt
# 导入图像
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'未读取到照片, 检查路径后重试')
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

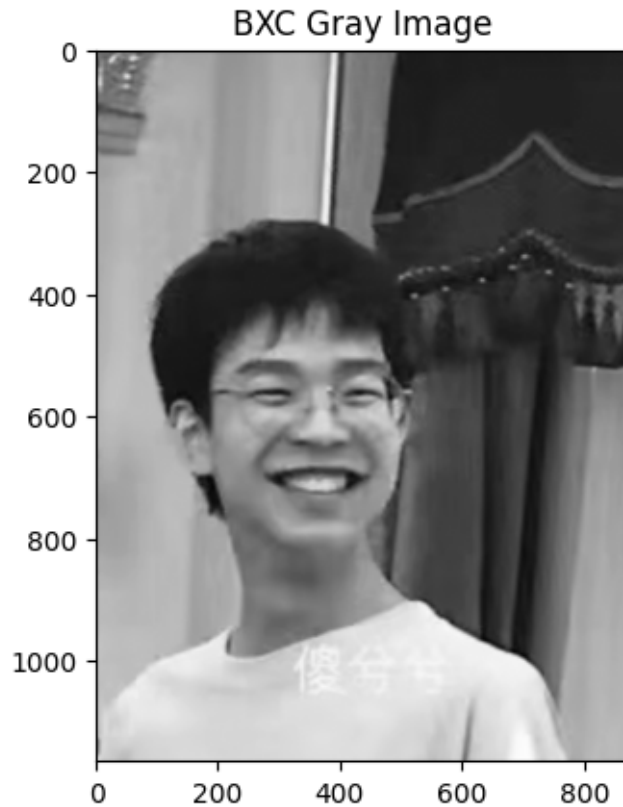


事实上，本部分依托于 opencv-python 库调用 cv2 十分简单。代码如下所示。

```
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # BGR—>GRAY
```

```
# 灰度图像转换 (plt 导出)
import cv2
import matplotlib.pyplot as plt
# 导入图像
img_path = '/Users/guo2006/myenv/Machine Vision Experiment/bxc.jpg'
img_bxc = cv2.imread(img_path)
if img_bxc is None:
    raise FileNotFoundError(f'未读取到照片，检查路径后重试')
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()
```



## 2.4 图像变换操作

此部分包含了图像的缩放操作、旋转操作及剪裁操作。下面将分步骤进行展开。

### 2.4.1 图像缩放操作

图像缩放操作主要包含两种类型：指定横纵长宽度放缩、指定缩放比例放缩。

```
# 缩放操作
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)) # 指定 pixel 大小

scale = 0.1 # 指定缩放比例
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')

```

对于两种不同情况的放缩方式，分别进行阐释：

- 指定横纵长宽度放缩（图像比例可能改变）

```
img_new = cv2.resize(img, (Width, Height))
```

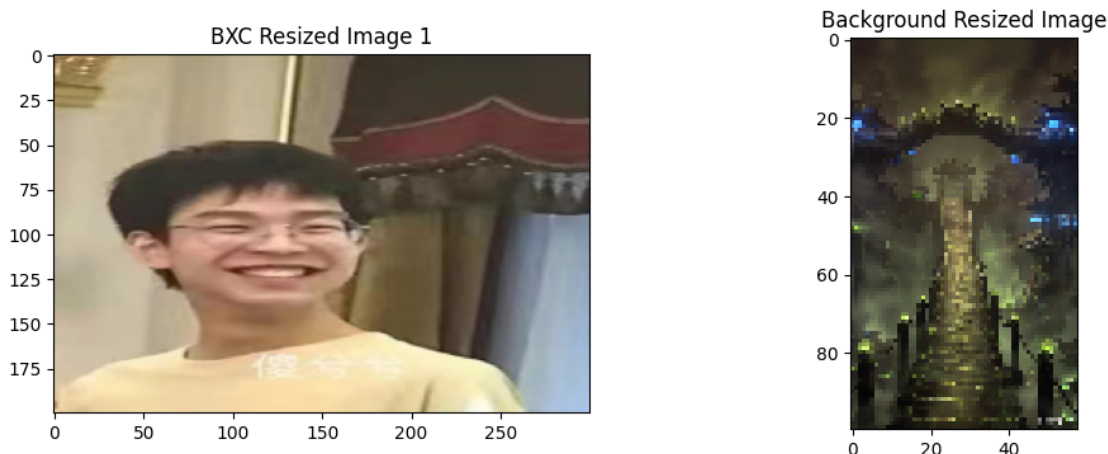
- 指定缩放比例放缩（图像比例不会改变）

```

scale = 0.1 # 指定缩放比例
img_new = cv2.resize(img, None, fx = scale, fy = scale)

```

```
Text(0.5, 1.0, 'Background Resized Image')
```



### 2.4.2 图像旋转操作

本质上，图像旋转操作并不难，但是图像在旋转后可能会有因画布大小不足而四角无法显示全的问题。故如何实现画布自适应大小，确保旋转后图像能够被显示完全是旋转操作最大的难点。我们所需要解决的问题无非是绕哪旋转、如何旋转、旋转多少度、是否缩放等问题。

- **绕哪旋转**：要绕中心旋转，先要得到中心坐标。即是有：

$$Rotation\ Center = (X, Y) = (\frac{Width}{2}, \frac{Height}{2})$$

根据 `img.shape` 进行切片得到 `Width` 和 `Height` 即可。

- **旋转多少度**：给定一个参数 `Rotation Angle` 即可，在后续带入 `cv2` 函数。
- **缩放比例**：给定一个参数 `Scale` 即可，后续带入 `cv2` 函数。

在简单旋转的情况下，也就是不考虑四角会因画布大小不足被裁切的情况下，可以直接调用 `cv2.getRotationMatrix2D` 函数。如下所示：

```
M = cv2.getRotationMatrix2D(Rotation Center, Rotation Angle, Scale)
img_rotated = cv2.warpAffine(img, M, (Width,Height)) # 更新后的矩阵重采样
```

```
# 旋转操作 (难点)
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'未找到对应文件，检查路径后重试')
```

```

else:
    img_before = cv2.cvtColor(img_before, cv2.COLOR_BGR2RGB)
    height, width = img_before.shape[:2] # 对.shape 返回的三数数组进行切片, 通道数
    # 值舍弃

    center = (width//2, height//2) # 找到对应旋转中心
    rotation_angle = 30 # 逆时针旋转 30deg
    scale = 1 # 缩放系数

    # 计算旋转矩阵
    M = cv2.getRotationMatrix2D(center, rotation_angle, scale)
    #img_before_rotated = cv2.warpAffine(img_before, M, (width,height)) # 更新后
    # 的矩阵重采样

    #cv2.imshow('<before> Rotated Picture',img_before_rotated)
    #cv2.waitKey(0)
    #cv2.destroyAllWindows() # 显示出的四个角由于画布大小不够将会被切掉

    # 自动调整画布大小
    cos, sin = abs(M[0,0]), abs(M[0,1])
    new_width = int(height*sin + width*cos)
    new_height = int(width*sin + height*cos)
    # 补齐偏移量
    M[0,2] += new_width/2 - center[0] #center[0] 是 width//2
    M[1,2] += new_height/2 -center[1] #center[1] 是 height//2
    # 用更新后的矩阵 M 和扩大后的画布大小 (new_width,new_height) 做重采样
    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()

```

在考虑复杂旋转情况下, 需要一定的数学变换来进行画布自适应大小的调节。具体数学原理如下:  
cv2.getRotationMatrix2D 返回一个  $2 \times 3$  的仿射阵, 具体如下:

$$M = cv2.getRotationMatrix2D = \begin{bmatrix} \cos\theta & -\sin\theta & tx \\ \sin\theta & \cos\theta & ty \end{bmatrix}$$

其中， $\theta$  为旋转角度，tx、ty 为偏移量。

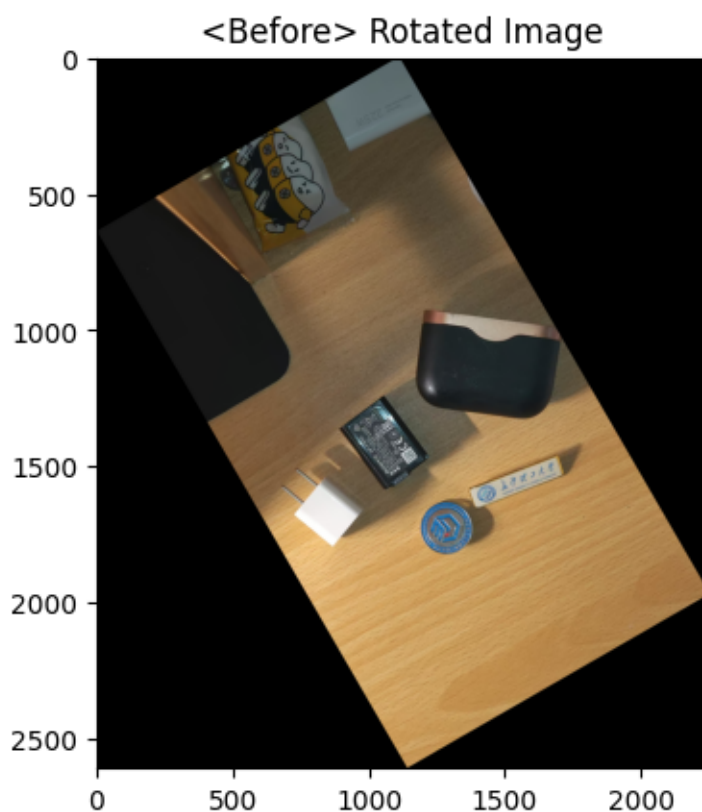
旋转后所需的最小宽度  $Width_{min} = Height_{Original} \times |\sin\theta| + Width_{Original} \times |\cos\theta|$

旋转后所需的最小高度  $Height_{min} = Width_{Original} \times |\sin\theta| + Height_{Original} \times |\cos\theta|$

画布中心有偏移，要通过 tx、ty 补齐偏移量。具体如下：

原图中心  $(\frac{Width}{2}, \frac{Height}{2})$  ——> 新画布中心  $(\frac{Width_{new}}{2}, \frac{Height_{new}}{2})$

- $tx = M_{0,2} + \frac{Width_{new}}{2} - center[0]$
- $ty = M_{1,2} + \frac{Height_{new}}{2} - center[1]$



### 2.4.3 图像剪裁操作

给出变量存储行范围与列范围，对指定范围进行裁剪。

```
# 剪裁操作
import cv2
```



```

import matplotlib.pyplot as plt

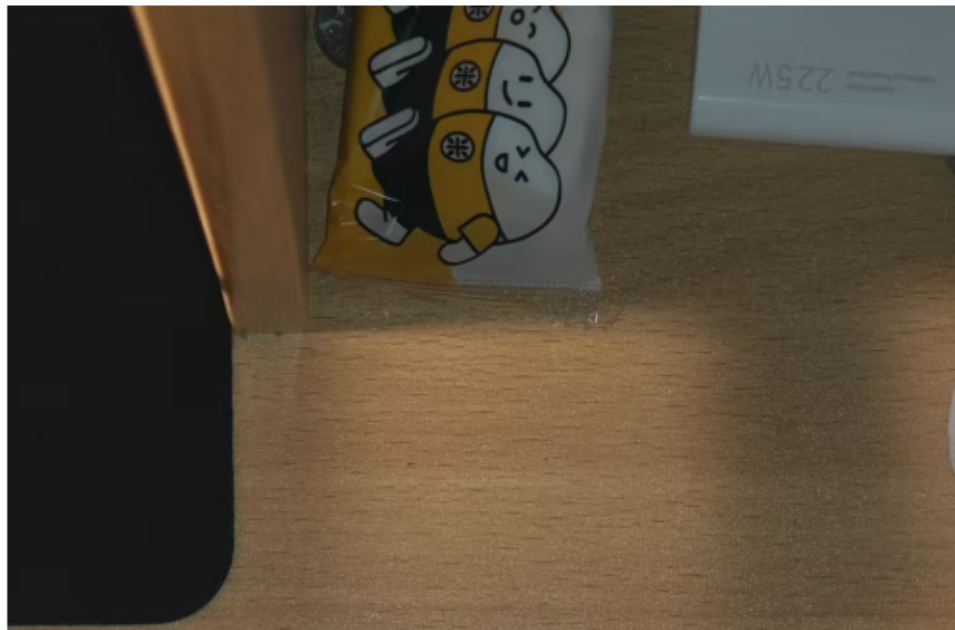
img_path = '/Users/guo2006/myenv/Machine Vision Experiment/after.jpg'
img_after = cv2.imread(img_path)
if img_after is None:
    raise FileNotFoundError(f'未找到对应文件，检查路径后重试')
else:
    img_after = cv2.cvtColor(img_after, cv2.COLOR_BGR2RGB)
    y1, y2 = 50, 820 # 行范围
    x1, x2 = 115, 1530 # 列范围

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

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

```

<After> Cropped Image



给出四变量存储行范围列范围，再对目标进行切片即可。

## 2.5 图像运算操作

图像运算操作包括加法运算、减法运算、乘法运算、除法运算四种。

值得注意的是，进行运算的图像必须要将图片大小统一。需要运用相似数学原理。

设原图高度为  $h_0$ ，原图宽度为  $l_0$ ；修改后的图像高度为  $h$ ，宽度为  $l$ 。

令改后的高度等于所传入的第一张图的高度，宽度根据相似原理进行计算。

则有：

$$\frac{h}{h_0} = \frac{l}{l_0}$$
$$l = \frac{h \times l_0}{h_0}$$

下面将详细阐释各种运算方法。

### 2.5.1 加法运算

加法运算包含平均降噪和双重曝光两种，下面将分别进行阐释。

```
# 图像加法运算-> 平均降噪
import cv2
import matplotlib.pyplot as plt
import numpy as np
import os
import random

# 显示函数（更简便地展示多图对比）
def show(title, *imgs):
    # 多张图片水平拼成一行，同一高度后显示
    h0 = imgs[0].shape[0] # 传入的第一张图为统一高度

    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()
# 图像平均函数 (降噪)
def add_noise(img,sigma=25):
    # 添加高斯噪声
    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 # 生成 50 张噪声图像

noise_img_list = [] # 创建随机噪声图像集
for _ in range(N):
    noise_img_list.append(add_noise(img_bxc_original))
# 求噪声抑制的平均图像
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)

```

Addition——Image Denoising



```

# 图像加法运算-> 双重曝光
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)
img_background = cv2.imread(img_background_path)
# 色彩模式转换
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]
# 调整背景图片与主体图片尺寸一致
img_background = cv2.resize(img_background,(width_bxc,height_bxc))
# 双重曝光加权
double_exposure = cv2.addWeighted(img_bxc,0.6,img_background,0.4,0)

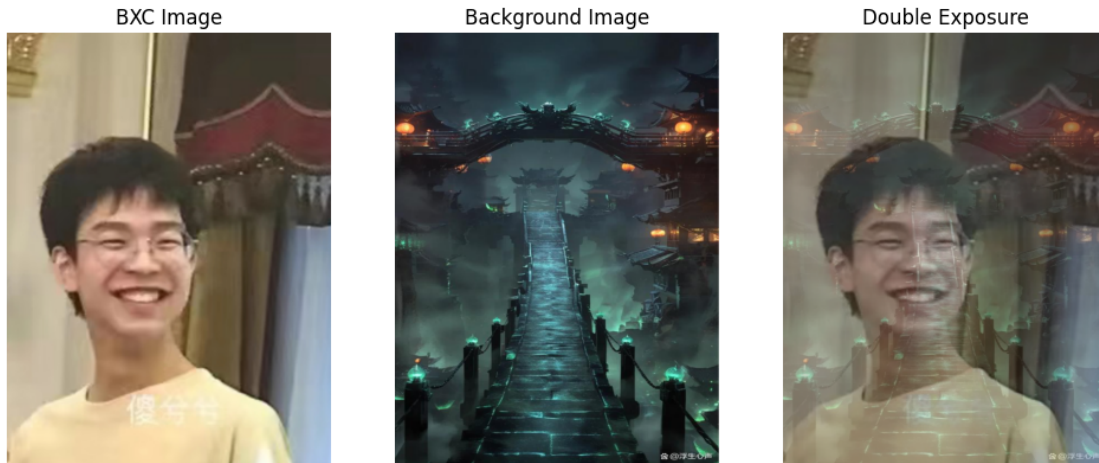
fig, ax = plt.subplots(1,3,figsize=(12,5))
ax[0].imshow(img_bxc)
ax[0].set_title('BXC Image')
ax[0].axis('off')
ax[1].imshow(img_background)
ax[1].set_title('Background Image')
ax[1].axis('off')
ax[2].imshow(double_exposure)
ax[2].set_title('Double Exposure')
ax[2].axis('off')

```

```

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

```



### 2.5.2 减法运算

减法运算主要运用于静物识别，其可以将两张相同视角的不同静物照片进行相减，显示出高光部分为差异区。减去的部分为黑色阴暗区。

利用大津算法来寻找最佳二值化值，并输出最佳二值化阈值。大津二值化法用来自动对基于聚类的图像进行二值化，或者说，将一个灰度图像退化为二值图像。

```
# 减法 -> 静物差异识别
import cv2
import matplotlib.pyplot as plt
import numpy as np
import os
import random

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)

# 色彩模式转换
img_before = cv2.cvtColor(img_before, cv2.COLOR_BGR2RGB)
img_after = cv2.cvtColor(img_after, cv2.COLOR_BGR2RGB)
# 修改为统一尺寸
height_before, width_before = img_before.shape[:2]
img_after = cv2.resize(img_after, (width_before, height_before))
# 做差值图
img_diff_abs = cv2.absdiff(img_before, img_after)
```

```

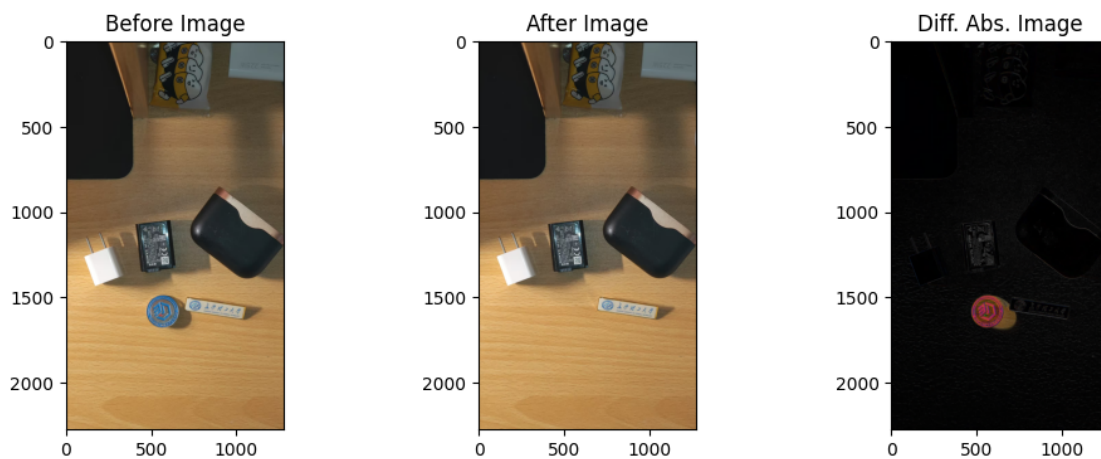
img_diff_abs_gray = cv2.cvtColor(img_diff_abs, cv2.COLOR_BGR2GRAY)
# 调用大津算法计算最佳阈值
thresh_used, mask = cv2.threshold(img_diff_abs_gray,0,255,cv2.THRESH_BINARY +
    ↪cv2.THRESH_OTSU)
print('Otsu Automatic Threshold=',thresh_used)

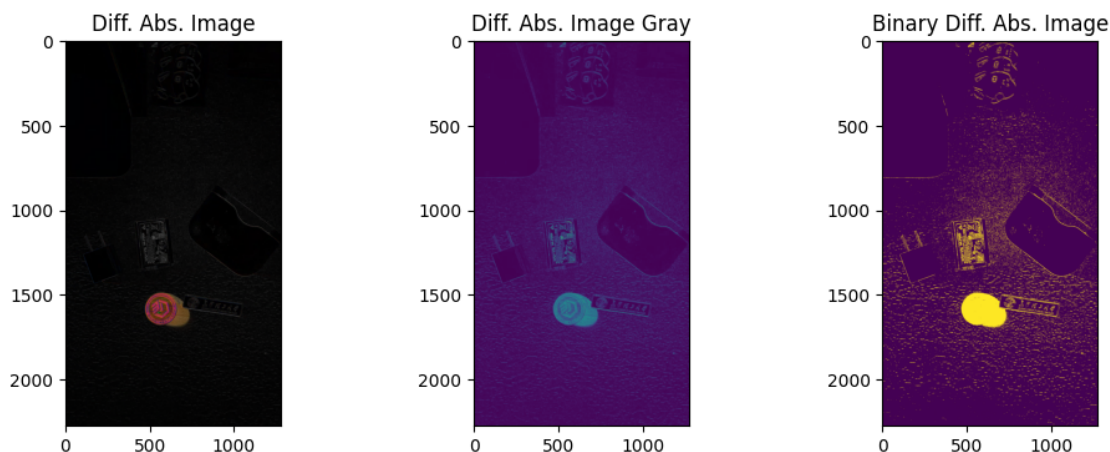
fig, ax = plt.subplots(1,3,figsize=(12,4))
ax[0].imshow(img_before)
ax[0].set_title('Before Image')
ax[1].imshow(img_after)
ax[1].set_title('After Image')
ax[2].imshow(img_diff_abs)
ax[2].set_title('Diff. Abs. Image')
fig, ax = plt.subplots(1,3,figsize=(12,4))
ax[0].imshow(img_diff_abs)
ax[0].set_title('Diff. 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= 22.0

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





### 2.5.3 乘法运算

图像乘法是将两幅图像的对应像素值相乘，生成一幅新的图像。

乘法运算有许多用途，比如我们可以为各个像素乘上一个系数，整体提亮或减暗张图片的某个局部或整体；还可以单独提取出来图像的某一个部分。但是无论哪种用法，我们都要先为图像创建一张‘掩膜’(Mask)。它相当于量化照片的一种规则或模版。掩膜的系数不同，对图像有不同的作用。

图像的乘法运算主要包含两种类型：

- 二值掩膜  $<0 / 255>$
- 灰度掩膜  $<0 \sim 255>$

与其说乘法运算有两种类型，不如说掩膜的量化模型有两种。不过，不同的类型起到的作用不相同，具体要看情况分析。

```
# 乘法（掩膜抠图）
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]
# 二值掩膜
mask_bin = np.zeros((height_bxc,width_bxc),np.uint8) # 创建黑色掩膜
```



```

cv2.circle(mask_bin,(width_bxc//2,height_bxc//2),min(height_bxc, width_bxc)//3,
↳255, -1) # 使黑色掩膜的中间部分出现一个白色的圆域

result_bin = cv2.bitwise_and(img_bxc, img_bxc, mask=mask_bin) # 掩膜内白色圆域内
内容留下，其余部分删除
# 灰度掩膜
Y, X = np.ogrid[:height_bxc, :width_bxc] # 创建欧式坐标网格
dist = np.sqrt((X-width_bxc)**2+(Y-height_bxc)**2) # 计算各个像素距离中心的欧氏距
离
# 把距离线性映射到 0-255 的灰度（设定掩膜规则）
mask_gray = np.clip((255-dist*(255/min(height_bxc, width_bxc)/2)),0,255).
↳astype(np.uint8)
# 可能导致灰度变化不明显？ 怎么解决
result_gray = cv2.bitwise_and(img_bxc, img_bxc, mask=mask_gray)

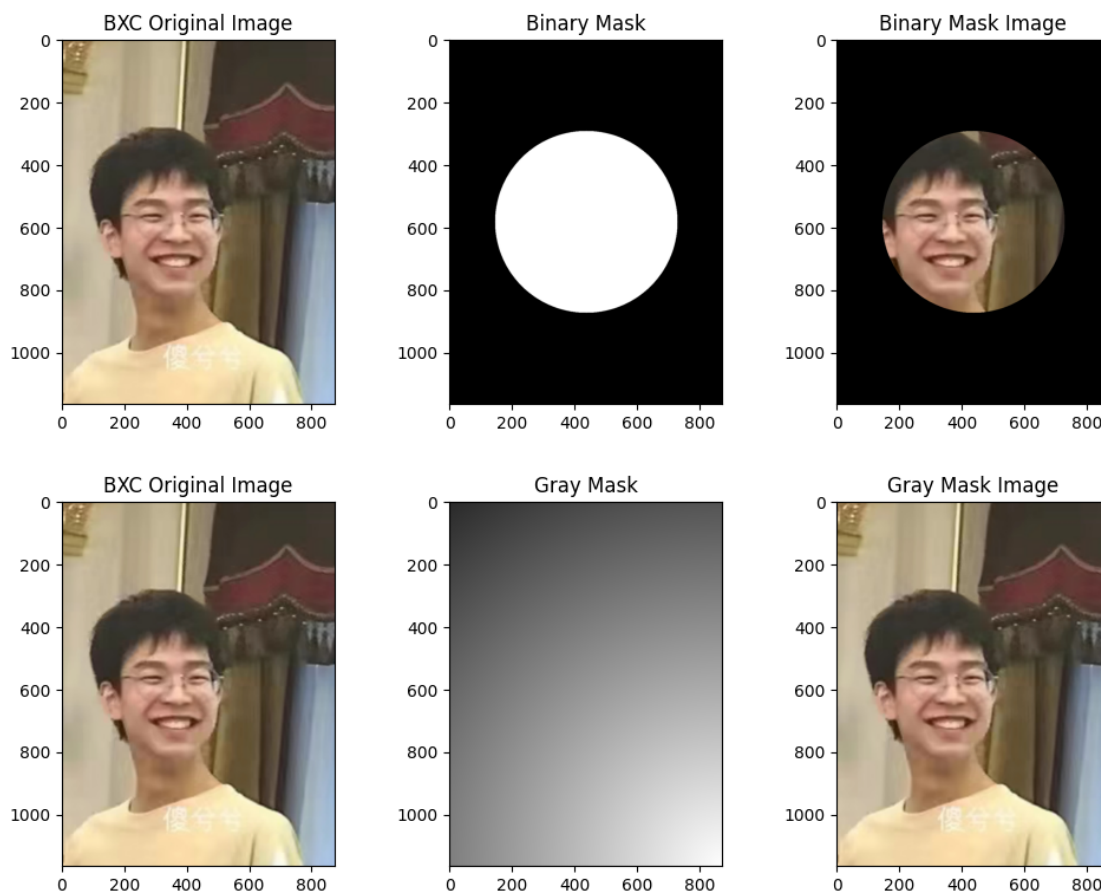
img_bxc = cv2.cvtColor(img_bxc, cv2.COLOR_BGR2RGB)
mask_bin = cv2.cvtColor(mask_bin, cv2.COLOR_BGR2RGB)
mask_gray = cv2.cvtColor(mask_gray, cv2.COLOR_BGR2RGB)
result_bin = cv2.cvtColor(result_bin, cv2.COLOR_BGR2RGB)
result_gray = cv2.cvtColor(result_gray, cv2.COLOR_BGR2RGB)

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')

```



Text(0.5, 1.0, 'Gray Mask Image')



## 2.5.4 除法运算

图像除法是两幅图像的对应像素值相除生成新图像的过程，其包含了两种不同的模型：

- 常数除法——> 均匀的将整张图片变暗
- 非常数除法——> 抵消光照不均匀 用于图像矫正

本质上两者的不同也是在掩膜层面上的，事实上，原图与掩膜层在光照参数上的叠加便是处理后的图像。在这一点上，乘法与除法是有共同之处的。

```
# 除法
import cv2
import matplotlib.pyplot as plt
import random
import os
import numpy as np
```

```

# 常数除法---> 把整张图均匀变暗
img_bxc_path = '/Users/guo2006/myenv/Machine Vision Experiment/bxc.jpg'
img_bxc = cv2.imread(img_bxc_path).astype(np.uint8) # 转换成 0—255 的浮点数，方便做除法

img_bxc_dark = (img_bxc/3).clip(0,255).astype(np.uint8) # 除整 + 截断 + 转换
img_bxc_dark = cv2.cvtColor(img_bxc_dark, cv2.COLOR_BGR2RGB)
# 非整除法---> 抵消光照不均 (图像矫正)
height_bxc, width_bxc = img_bxc.shape[:2]
Y, X = np.ogrid[:height_bxc, :width_bxc] # 创建网格

light_illum = np.sqrt((X-width_bxc)**2+(Y-height_bxc)**2) # 距离图片中心欧式距离
light_illum = (light_illum/light_illum.max()*0.8+0.2).astype(np.float32) # 归一化到 0.2~1.0
#light_illum 为双通道，img_bxc 为三通道，二者无法用 np 相除，办法有两种：
#1.将 img_bxc 转换为 img_bxc_gray(BGR->GRAY)，两个双通道可相除 < 双通道灰度图 >
#2.将 light_illum 转换为三通道 < 三通道彩图 >

# 法 1.双通道灰度图像
img_bxc_gray = cv2.cvtColor(img_bxc, cv2.COLOR_BGR2GRAY)
# 除法抵消暗区 + 0~255 截断 + 格式转换
corrected_img_2ch = (img_bxc_gray / light_illum).clip(0,255).astype(np.uint8)

# 法 2.三通道彩色图像
# 扩充光线模版为三通道
light_illum_3ch = cv2.merge([light_illum, light_illum, light_illum])
# 除法抵消暗区 + 0~255 截断 + 格式转换
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)

```

```

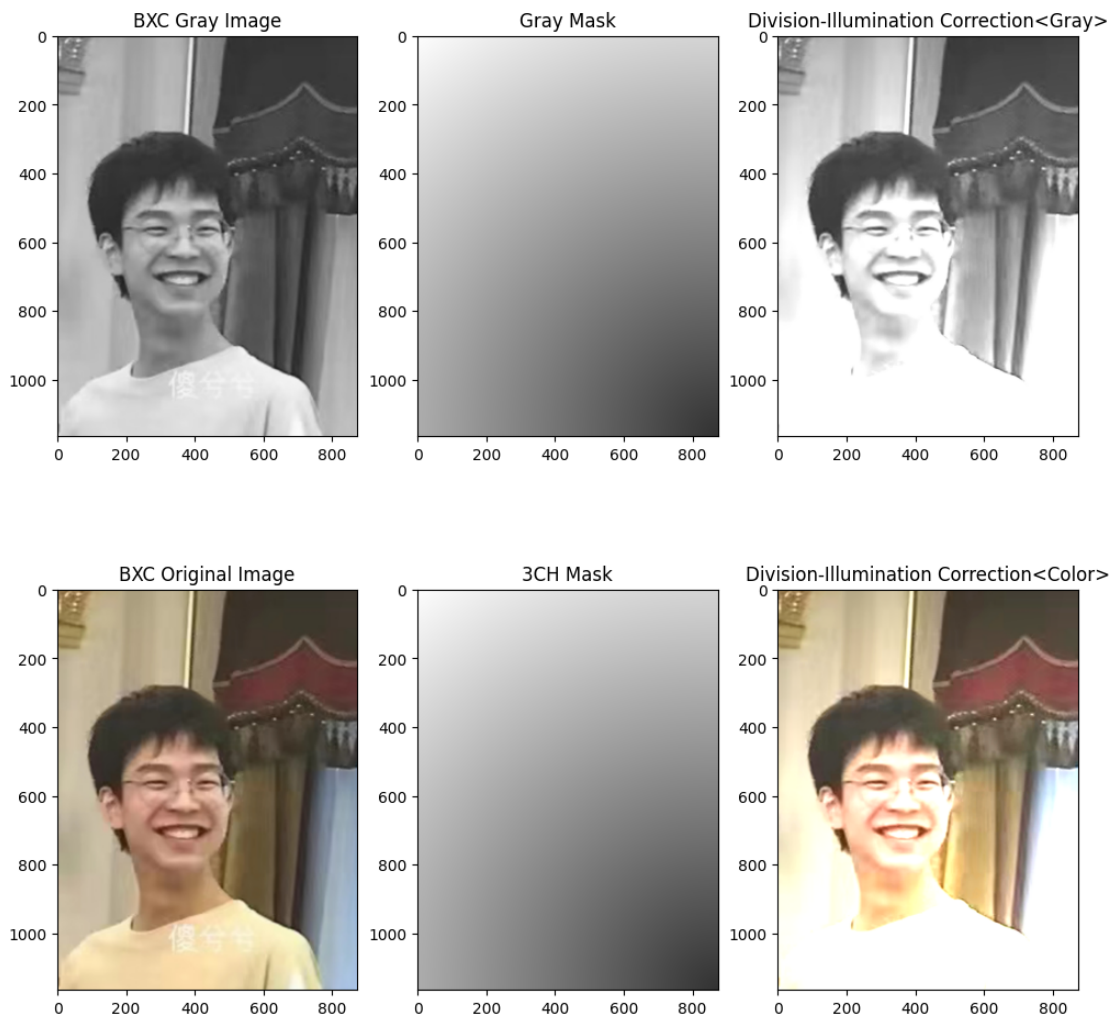
fig, ax = plt.subplots(2,3,figsize=(12,12))
ax[0,0].imshow(img_bxc_gray)
ax[0,0].set_title('BXC Gray Image')
ax[0,1].imshow((light_illum*255).astype(np.uint8))
ax[0,1].set_title('Gray Mask')
ax[0,2].imshow(corrected_img_2ch)
ax[0,2].set_title('Division-Illumination Correction<Gray>')
ax[1,0].imshow(img_bxc)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

```

```

Text(0.5, 1.0, 'Division-Illumination Correction<Color>')

```



## 2.6 逻辑运算

一个图像可以用像素矩阵来表示，通过随机数生成一个新的像素矩阵，称之为密钥图像，将原图与密钥图像进行按位异或运算，对图像进行加密，生成一个加密图像，这是对图像的整体打码。通常通过掩膜提取图像中感兴趣区域（ROI），再对 ROI 进行加密，这是对图像的局部打码。将加密图像与原密钥图像按位异或运算，进行解密，可以得到原图，这个过程叫做解码。

本小节将使用 opencv-python 库自带的 haar 人脸识别系统，对人脸部分进行检测与识别、并进行随机密钥打码操作，然后再次使用密钥进行反运算来解密并显示。

```
#ROI、解码
import cv2
import matplotlib.pyplot as plt
import random
import os
```

```

import numpy as np

# 加载人脸检测器
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)
if face_img is None:
    raise FileNotFoundError(f'图像路径错误，检查路径后重试')
face_img_gray = cv2.cvtColor(face_img, cv2.COLOR_BGR2GRAY)
# 检测脸部 -> 得到 ROI 掩膜
faces = face_cascade.detectMultiScale(face_img_gray, scaleFactor=1.2,
    ↪minNeighbors=5)
# 返回值 faces 是一个 N×4 的 numpy 数组，每行 (x, y, w, h) 代表一张人脸的左上角坐标和宽高。
# 创建全黑掩膜
mask = np.zeros(face_img.shape[:2], dtype=np.uint8) # 单通道全黑
# 脸部涂白
for (x, y, w, h) in faces:
    cv2.rectangle(mask, (x, y), (x+w, y+h), 255, -1)
    # 绘制在 mask 上，左上角，右下角，255= 白，实心填充
# 生成随机密钥
random_key = np.random.randint(0, 256, face_img.shape, dtype=np.uint8) # 0-255 随机
# 生成的尺寸与 face_img 完全一样的三位数组
# 只对人脸区域 (mask>0 区域) 进行异或加密
encrypted = face_img.copy() # 先整图复制一份，后面只改人脸部分，背景不动
encrypted[mask > 0] = cv2.bitwise_xor(face_img, random_key)[mask > 0]

# 再进行一次异或加密进行解密
decrypted = encrypted.copy()
decrypted[mask > 0] = cv2.bitwise_xor(encrypted, random_key)[mask > 0]

# 色彩转换
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>')
```

```
Text(0.5, 1.0, 'Decrypted<Face ROI>')
```



## 3 L<sup>A</sup>T<sub>E</sub>X 使用基础命令

### 3.1 代码基本结构

```
\documentclass{article} % 定义文档类型
\usepackage{amsmath, amssymb, geometry, graphicx, float, caption, subfigure,
booktabs, tcolorbox, ctex} % 引入宏包
\geometry{a4paper, margin=2.5cm} % 设置页边距

\title{标题} % 设置标题
\author{作者名 1 \thanks{通讯作者.} \and 作者名 2 \and etc.} % 设置作者
\date{日期} % 设置日期

% 正式结构 %
\begin{document}
  \maketitle % 忘加此句会导致无标题!

  \begin{abstract}
    摘要部分内容. % 填写摘要内容
  \end{abstract}

  \tableofcontents % 自动创建目录
  \newpage % 换页命令

  \section{第一部分-标题}
    第一部分-正文
  \subsection{第一部分 第一章-标题}
    第一部分 第一章-正文
  \subsubsection{第一部分 第一章 第一节-标题}
    第一部分 第一章 第一节-正文
\end{document}
```

\* 注：此处 `\usepackage` 使用的宏包已足够  $\leq 90\%$  的排版场景，在后续表3中将标注各宏包的功能。

上面的代码框即为基本 L<sup>A</sup>T<sub>E</sub>X 文档结构，掌握基本结构后即可在基本结构的基础上学习细节内容，填充在结构中就可以形成一篇文章了。本章接下来的内容里，将会分类介绍、由简入繁地介绍各个部分的书写办法。

基本步骤为：公式 → 枚举 → 图 → 表 → 文本框 → 引用文献

宏包名	功能
amsmath amssymb	AMS 数学拓展包 数学拓展包
geometry float multicol	页面设置 图表位置设置 单双栏页面编辑
graphicx caption subfigure booktabs tcolorbox makecell	插图包 图表标题 多图拓展包 三线表 文本框拓展包 表格内换行

### 3.2 单双栏版式变换

这节的出现比较矛盾，它只是一个点，但是却又有时候很重要。但是将它放到以下任一节都略显突兀、格格不入。故此处单独拿出一节来说明下它。

这个地方我们将会用到 `multicol` 宏包，我将其译为多样栏功能。使用此包你可以既在某些地方使用一栏版式，又可以在某些地方分成两栏、三栏乃至更多。其基础语法很简单，只需在你想要分栏的地方加入 `multicols` 环境即可。演示如下。

```
\begin{document}
    对于此处不在 multicols 环境内的内容，都是 单栏 内容。
    \begin{multicols}{2}
        对于此在 multicols 环境内的内容，LaTeX 将会自动分为 两栏 显示。
    \end{multicols}
\end{document}
```

对于此处不在 `multicols` 环境内的内容，都是单栏内容。

*LaTeX* 对于此处在 `multicols` 环境内的内容，其将会自动把 `multicols` 环境内的内容分为两栏显示。

总结 `multicols` 语法为：

```
\begin{multicols}{分栏数目}
    分栏内容
\end{multicols}
```



### 3.3 公式插入

在此节，将讲述普通公式的插入、公式作为转义字符在文本插入领域的应用等。由于矩阵的插入具有一定特殊性，故单独拿出一小节梳理。

#### 3.3.1 普通公式

对于普通公式，大致要考虑三个问题：

- 行内公式 or 行间公式
- 编号公式 or 不编号公式
- 单行公式 or 多行公式

可以简单画如下表4相应的表格，使得三个问题更加直观。

分类	单行公式	多行公式
行内公式	不编号公式 ( <i>Con.1</i> )	——
行间公式	编号公式 ( <i>Con.2</i> )	编号公式 ( <i>Con.4</i> )
	不编号公式 ( <i>Con.3</i> )	不编号公式 ( <i>Con.5</i> )

- 对于 *Con.1* →

`$ Equation $` **%Equation** 处为输入公式

- 对于 *Con.2* →

```
\begin{equation}
Equation %Equation 处为输入公式
\end{equation}
```

- 对于 *Con.3* →

```
\begin{equation*} % 只需加入 * 号即可不编号公式.
Equation %Equation 处为输入公式
\end{equation*}
```

另外，还有一种方式可以不编号行间公式。如下所示。

`$$Equation$$`

- 对于 *Con.4* →

```
\begin{align}
Equation1 \\\
Equation2 \\\
使用 &= 可以在等号处对齐.
\end{align}
```

- 对于 *Con.5* →

```
\begin*{align} % 只需加入 * 号即可不编号此部分的全部公式.
  \underline{Equation1} \\
  \underline{Equation2} \\
  \underline{使用 \&= 可以在等号处对齐.}
\end{align}
```

此外, 还可以使用多个  $\underline{Equation}$  换行叠加来实现。如下所示.

```
$$\underline{Equation1}$$ \\
$$\underline{Equation2}$$
```

### 注意!

若只想要对多行行间公式的部分公式不编号, 可以采用 `\notag` 对其进行处理。如下所示.

```
\begin{align}
  \underline{Equation1} \\
  \underline{Equation2} \notag \\ % 此行公式不会被编号.
  \underline{Equation3} \\
  \underline{使用 \&= 可以在等号处对齐.}
\end{align}
```

对于公式输入这一模块, 除了掌握 (记忆) 基本的运算符之外, 还应当多加练习。L<sup>A</sup>T<sub>E</sub>X 公式的符号有很多, 下面列举部分常用的 L<sup>A</sup>T<sub>E</sub>X 公式符号. (如下表5所示)

除此之外, 关于换行与空格的知识如下。

#### 1. 空格符号

- 特殊空格符号 `~` → 不间断空格符, 通常用于避免在两行之间断开。
- `\+ Space 键位` → 较短的不间断空格, 可以用在需要较小空格的位置。
- `\hspace{长度}` → 插入一个特定长度的空格。  
单位: **em** (字体大小单位) / **cm** (厘米) / **pt** (磅) 等。  
\* 注: `\hspace*{长度}` → 强制插入一个特定长度的空格 (在行首也会插入)
- `\quad` 及 `\qquad` → 简便空格命令, 分别插入 1em 和 2em 宽度的空格。

#### 2. 换行符号

- `\vspace{长度}` → 插入指定长度的垂直空白。

第一行内容..... \\

\vspace{1cm}

第二行内容.....

\* 注: `\vspace*{长度}` → 强制插入一个特定长度的垂直空白（在页面顶部也会插入），可以用于页面顶部的排版调整。

- `\\` → 换行符，常用于表格、公式以及某些环境中强制换行。
- `\newline` → 与 `\\` 类似，但在某些情况下更符合语法规范。

第一行内容..... \newline

第二行内容.....

### 3. 其余功能

- `\\*` → 换行并禁止分页，适用于需要将两行内容固定在同一页的情况。
- **par 环境** → 在 `par` 环境中的内容会自动根据页面宽带换行。

符号	语法	符号	语法	符号	语法	符号	语法
$\alpha$	<code>\alpha</code>	$\beta$	<code>\beta</code>	$\gamma$	<code>\gamma</code>	$\theta$	<code>\theta</code>
$\varepsilon$	<code>\varepsilon</code>	$\delta$	<code>\delta</code>	$\mu$	<code>\mu</code>	$\nu$	<code>\nu</code>
$\eta$	<code>\eta</code>	$\zeta$	<code>\zeta</code>	$\lambda$	<code>\lambda</code>	$\psi$	<code>\psi</code>
$\sigma$	<code>\sigma</code>	$\xi$	<code>\xi</code>	$\tau$	<code>\tau</code>	$\phi$	<code>\phi</code>
$\varphi$	<code>\varphi</code>	$\rho$	<code>\rho</code>	$\chi$	<code>\chi</code>	$\omega$	<code>\omega</code>
$\pi$	<code>\pi</code>						
$\Sigma$	<code>\Sigma</code>	$\Pi$	<code>\Pi</code>	$\Delta$	<code>\Delta</code>	$\Gamma$	<code>\Gamma</code>
$\Psi$	<code>\Psi</code>	$\Theta$	<code>\Theta</code>	$\Lambda$	<code>\Lambda</code>	$\Omega$	<code>\Omega</code>
$\Phi$	<code>\Phi</code>	$\Xi$	<code>\Xi</code>				
$+$	<code>+</code>	$-$	<code>-</code>	$\times$	<code>\times</code>	$\div$	<code>\div</code>
$\partial$	<code>\partial</code>	$\infty$	<code>\infty</code>	$\rightarrow$	<code>\rightarrow</code>	$\leftarrow$	<code>\leftarrow</code>
$\sum$	<code>\sum</code>	$\prod$	<code>\prod</code>	$\int$	<code>\int</code>	$\oint$	<code>\oint</code>
$\iint$	<code>\iint</code>	$\iiint$	<code>\iiint</code>	$\frac{a}{b}$	<code>\frac{a}{b}</code>	$\vec{a}$	<code>\vec{a}</code>
$\sqrt{x}$	<code>\sqrt{x}</code>	$\sqrt[n]{x}$	<code>\sqrt[n]{x}</code>	$\dot{a}$	<code>\dot{a}</code>	$\ddot{a}$	<code>\ddot{a}</code>
$f'(x)$	<code>{f}'(x)</code>	$f''(x)$	<code>{f}''(x)</code>	$x^n$	<code>x^{n}</code>	$x_n$	<code>x_{n}</code>

### 3.3.2 矩阵

矩阵环境分为 `pmatrix`、`bmatrix`、`Bmatrix`、`vmatrix`、`Vmatrix`、`matrix` 六种，接下来将逐步展开说明各种形式的矩阵形状。对于所有环境的矩阵，一大共性是 `&` 表示分割元素，而 `\\` 表示换行。

#### 1. `pmatrix` 环境下的矩阵

```
A=
\begin{pmatrix}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{pmatrix}
```

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

#### 2. `bmatrix` 环境下的矩阵

```
A=
\begin{bmatrix}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{bmatrix}
```

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

#### 3. `Bmatrix` 环境下的矩阵

```
A=
\begin{Bmatrix}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{Bmatrix}
```

$$A = \begin{Bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{Bmatrix}$$

#### 4. `vmatrix` 环境下的矩阵

```
A=
\begin{vmatrix}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{vmatrix}
```

$$A = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}$$

#### 5. `Vmatrix` 环境下的矩阵

```
A=
\begin{Vmatrix}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{Vmatrix}
```

$$A = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}$$

## 6. **matrix** 环境下的矩阵

```
A=
\begin{matrix}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{matrix}
```

$$A = \begin{matrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{matrix}$$

\* 注：所有矩阵必须存在于数学环境下。（即 `equation*` 或 `equation` 环境）

### 3.3.3 转义符号

本小节较为简单. 大致将转义字符分为两类：

1. **基本类** → 基本的一些符号，如 `#`, `$`, `%`, `&`, `{`, `}`, `_` 等。

<code>#</code> , <code>\$</code> , <code>%</code> , <code>&amp;</code> , <code>{</code> , <code>}</code> , <code>_</code> → <code>\#</code> , <code>\\$</code> , <code>\%</code> , <code>\&amp;</code> , <code>\{</code> , <code>\}</code> , <code>\_</code> .
--

2. **特殊类** → 比较特殊的符号，如 `^`, `-`, `\` 等。

- `^` 符号 → `\^{}`

- `-` 符号 → `\-{}`

- `\` 符号 → `\backslash$` (公式转义) 或 `\verb| |` 文本转义

## 3.4 枚举

枚举主要分为两种，一种为不计数分条枚举，一种为计数分条枚举。

1. 不计数分条枚举

```
\begin{itemize}
\item 第一条内容
\item 第二条内容
...
\item 第 n 条内容
\end{itemize}
```

- |  |
|--|
| <ul style="list-style-type: none"> <li>• <u>第一条内容</u></li> <li>• <u>第二条内容</u></li> <li>... ..</li> <li>• <u>第 n 条内容</u></li> </ul> |
|--|

## 2. 计数分条列举

```
\begin{enumerate}
  \item 第一条内容
  \item 第二条内容
  ...
  \item 第  $n$  条内容
\end{enumerate}
```

```
(a) 第一条内容
(b) 第二条内容
...
(c) 第  $n$  条内容
```

### 3.5 图片插入

本小节介绍如何借助 `\graphicx` 插入单图，以及借助 `\graphicx \subfigure` 库插入多图。共通之处是都需要借助一个 `\begin{figure}` 环境。

#### 3.5.1 单图插入

单图插入语法较为简单，如下所示。

```
\begin{figure} [H]      % 此处 H 为 \float 包中固定图片位置的指令
  \centering           % 表示使图片居中
  \includegraphics[ $width=0.5\text{width}$ ]{图片路径名}
  % 此处 width 表示 0.5 倍文字宽度，此外还有 scale= 缩放倍数、height= 图片高度 等指令。
  % 比如 [scale=0.75] 指的是缩放为原图比例的 0.75 倍，[height=5cm] 则指图片高度为 5cm。
  \caption{图名}
  \label{图名引用 ID}
\end{figure}
```

#### 3.5.2 多图插入

多图插入的语法在单图插入的基础上，加入 `subfigure` 语法即可，然后分别对图片进行定义。对分图片的定义同样有 `includegraphics`、`labe` 的步骤，但不一样的是，对于图片的标题，是在 `subfigure` 函数后以一 `[ ]` 号框起输入。示例语法如下所示。

```

\begin{figure}[H]
  \centering      % 使得图片居中
  \subfigure[图 A 图名]
    \label{图名引用 ID<A>}
    \includegraphics[width=0.5\textwidth]{图片路径名}
  \subfigure[图 B 图名]
    \label{图名引用 ID<B>}
    \includegraphics[width=0.5\textwidth]{图片路径名}
  \caption{总图名}
  \label{总图名引用 ID}
\end{figure}

```

### 3.6 表格插入

表格插入仅介绍两种常用的表格样式，一是常用的三线表，二是常用的正常框线的表格。

- 三线表

```

\begin{table}[H]
% 此处的 [H] 指令为固定位置 (float 宏包)
  \centering
  \caption{表名}
  \label{表名引用 ID}
  \setlength{\tabcolsep}{10pt}
% 列间距调整，默认 6pt.
  \begin{tabular}{@{}llc@{}}
% @{} 的作用为取消表格左右间距.
% llc 表示三列对齐方式 LeftLeftCenter
% 除了 [l],[c] 对齐方式外，还有 [r]→right.
    \toprule
      xx & xx & xx \\\
    \midrule
      xx & xx & xx \\\
      xx & xx & xx \\\
    \bottomrule
  \end{tabular}
\end{table}

```

行 1 列 1	行 1 列 2	行 1 列 3
行 2 列 1	行 2 列 2	行 2 列 3
行 3 列 1	行 3 列 2	行 3 列 3
行 4 列 1	行 4 列 2	行 4 列 3
行 5 列 1	行 5 列 2	行 5 列 3
行 6 列 1	行 6 列 2	行 6 列 3

需要注意的是，我们还可以通过在 llc 之间加入“|”符号，实现表格竖向线条的添加。这点在

全线表中会得到体现。

- 全线表

```
\begin{table}[H]
% 此处的 [H] 指令为固定位置 (float 宏包)
\centering
\caption{表名}
\label{表名引用 ID}
\setlength{\tabcolsep}{10pt}
% 列间距调整, 默认 6pt.
\begin{tabular}{@{}|l|l|c|@{}}
% @{} 的作用为取消表格左右间距.
% llc 表示三列对齐方式 LeftLeftCenter
% 除了 [l],[c] 对齐方式外, 还有 [r]→right.
\hline
xx & xx & xx \\\
\hline
xx & xx & xx \\\
xx & xx & xx \\\
\hline
\end{tabular}
\end{table}
```

行 1 列 1	行 1 列 2	行 1 列 3
行 2 列 1	行 2 列 2	行 2 列 3
行 3 列 1	行 3 列 2	行 3 列 3
行 4 列 1	行 4 列 2	行 4 列 3
行 5 列 1	行 5 列 2	行 5 列 3
行 6 列 1	行 6 列 2	行 6 列 3

## 3.7 文本框插入

### 3.7.1 基于 \fbox 简单文本框

- 单行文本框

```
\fbox{单行内容}
```

注意, 这种语法仅能为单行, 且仅能根据键入字符数目来创建对应长度的方框。  
且在文本过长时无法换行, 文本将溢出文本框。

- 多行文本框

```
\fbox{
\begin{minipage}{框宽}
正文内容, 可使用 \ 语法换行
\end{minipage}
% 需要新建一个 minipage(分页) 来实现此功能。
}
```



### 3.7.2 基于 `\tcolorbox` 复杂文本框

这种插入方法需要通过 `\` 语法进行换行。相关语法如下所示。

<div>标题 xxx(<i>colframe</i>)</div> <hr/> <div>正文 x(<i>colback</i>)</div>	<pre>\begin{tcolorbox}[colback=yellow!10,colframe=blue!50, title= 标题]     正文部分 \\ %! 后数字用来调节颜色透明度     正文第二行 \end{tcolorbox}</pre>
--	---

## 3.8 引用插入

### 3.8.1 正文引用

- 在正文中想要插入引用角标的地方加入 `\lable{引用 ID}`
- 然后在对应地方插入 `\ref{引用 ID}`

### 3.8.2 文献引用

- 在.tex 文件所在文件夹创建一.bib 文件，内需放置 BiBTeX 内容
- 在所需要正文引用的位置插入 `\cite{引用 ID}`
- 文章末尾引用文献，键入以下文本：

```
\bibliographystyle{nnsrt}
\bibliography{xxx(所创建的.bib 文件名)}
```