实验：图像位运算

实验概要

我们不再详细介绍二进制操作及其结果，下表提供了按位操作的真值表，以作为快速的复习：

<table>
<tr><th>Bitwise Operation</th><th>Table</th></tr>
<tr><td>NOT
Used for generating the negative of a binary image.
Function: <code>cv2.bitwise_not</code></td><td>
<table>
<tr><td>Input Bit</td><td>Output Bit</td></tr>
<tr><td>0</td><td>1</td></tr>
<tr><td>1</td><td>0</td></tr>
</table>
</td></tr>
<tr><td>OR
The OR operation will return a 1 if at least one of the images has a 1 in that pixel. This can be used to generate unions of two binary images.
Function: <code>cv2.bitwise_or</code></td><td>
<table>
<tr><td>Input Bit 1</td><td>Input Bit 2</td><td>Output Bit</td></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>
</td></tr>
<tr><td>AND
The AND operation will return a 1, but only if both of the images have a 1 in that specific pixel. This can be used to generate the intersection of two binary images.
Function: <code>cv2.bitwise_and</code></td><td>
<table>
<tr><td>Input Bit 1</td><td>Input Bit 2</td><td>Output Bit</td></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>
</td></tr>
<tr><td>XOR
The XOR operation will return a 1, but only if one of the pixels is 1 for the images. This can be used to identify the moving object in two subsequent frames.
Function: <code>cv2.bitwise_xor</code></td><td>
<table>
<tr><td>Input Bit 1</td><td>Input Bit 2</td><td>Output Bit</td></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>0</td></tr>
</table>
</td></tr>
</table>

下面直接进入实验操作。

实验目标

在本实验中，我们将使用 XOR（异或）操作，找出两个棋盘图像中被移动的棋子：

1. 导入依赖库

In [1]:

```
# 导入模块
import cv2                          # 导入OpenCV
import numpy as np                  # 导入NumPy
import matplotlib.pyplot as plt  # 导入matplotlib

# 魔法指令，使图像直接在Notebook中显示
%matplotlib inline
```

2. 加载图像

读取棋盘的图像，并将其转换为灰度

In [2]:

```
# 设置输入输出路径
import os
base_path = os.environ.get("BASE_PATH",'../data/')
data_path = os.path.join(base_path + "lab2/")
result_path = "result/"
os.makedirs(result_path, exist_ok=True)

# 读取第一幅棋盘的图像
img1 = cv2.imread("./data/board.png")
# 读取第二幅棋盘的图像
img2 = cv2.imread("./data/board2.png")
```

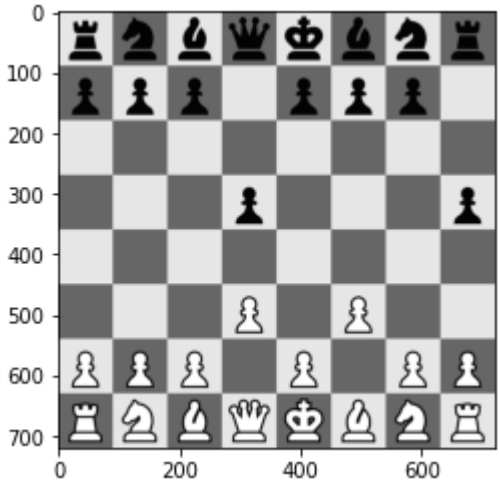
In [3]:

```
# 将第一幅棋盘的图像转换为灰度图片，共后续图像二值化操作
img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
# 将第二幅棋盘的图像转换为灰度图片，共后续图像二值化操作
img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY )
```

使用 Matplotlib 显示第一幅棋盘图像，输出信息如下。X 轴和 Y 轴分别为图像的宽度和高度:

In [4]:

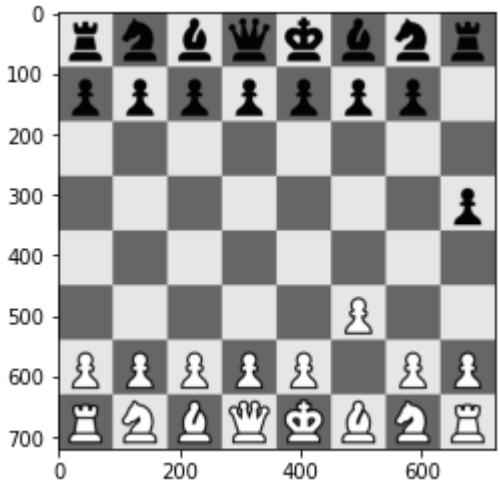
```
plt.imshow(img1,cmap="gray") # 使用灰色“喷涂”图像输出显示
plt.show()                   # 显示图像
```



使用 Matplotlib 显示第二幅棋盘图像，输出信息如下。X 轴和 Y 轴分别为图像的宽度和高度:

In [5]:

```
plt.imshow(img2,cmap="gray") # 使用灰色“喷涂”图像输出显示
plt.show()                   # 显示图像
```



3. 设置阈值和最大值

使用阈值 150 和最大值 255 对两个图像进行阈值处理:

In [6]:

```
# 设置阈（yu）值和最大值
# 设置阈值，小于等于阈值的像素将被替换为0
thresh = 150
# 设置最大值，大于阈值的像素将被替换为这里设置的最大值
maxValue = 255
```

4. 图像二值化

分别对两张棋盘图像： `img1` / `img2` 执行图像二值化；输出 `dit1` / `dst2`

In [7]:

```
# 执行图像二值化
th, dst1 = cv2.threshold(img1, thresh, maxValue, cv2.THRESH_BINARY)
```

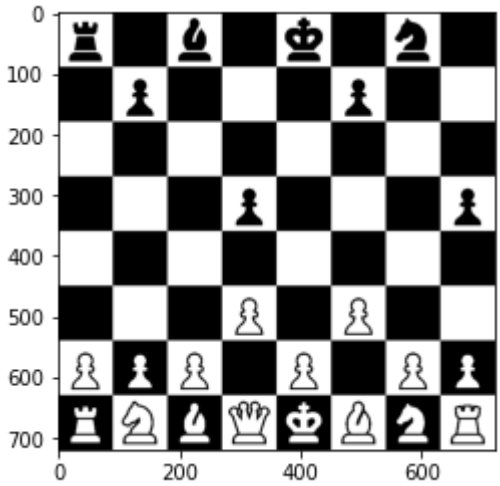
In [8]:

```
# 执行图像二值化
th, dst2 = cv2.threshold(img2, thresh, maxValue, cv2.THRESH_BINARY)
```

使用 Matplotlib 显示第一幅棋盘二值化后的图像，输出信息如下。X 轴和 Y 轴分别为图像的宽度和高度:

In [9]:

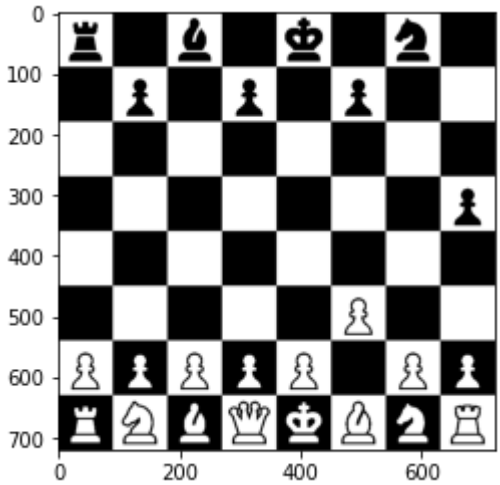
```
plt.imshow(dst1, cmap='gray') # 使用灰色“喷涂”图像输出显示
plt.show()                    # 显示图像
```



使用 Matplotlib 显示第二幅棋盘二值化后的图像，输出信息如下。X 轴和 Y 轴分别为图像的宽度和高度:

In [10]:

```
plt.imshow(dst2, cmap='gray') # 使用灰色“喷涂”图像输出显示
plt.show()                    # 显示图像
```



5. 按位执行与或运算

由于之前通过图像二值化，将两张棋盘图像转换成像素值非 0 即 1 的图片。因此，我们在这里可以使用 `cv2.bitwise_xor` 函数，执行按位 XOR ，查找已移动的片段。XOR 将逐位对比两个图像的像素值，如果两个二进制位相同，就返回 0，表示 false；否则返回 1，表示 true，从而将两幅二值图像的区别显示出来。

执行结果如下所示：

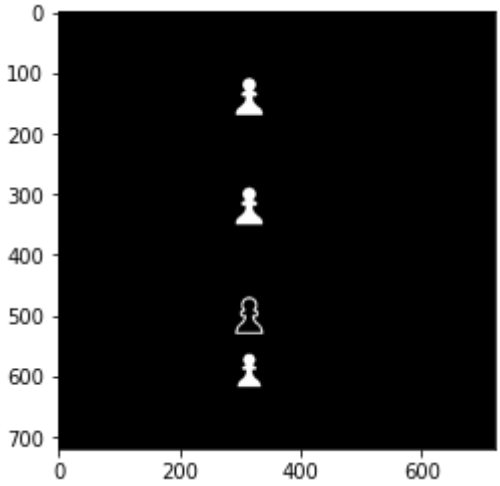
In [11]:

```
dst = cv2.bitwise_xor(dst1,dst2)
```

使用 Matplotlib 显示图像，输出信息如下。X 轴和 Y 轴分别为图像的宽度和高度:

In [12]:

```
plt.imshow(dst, cmap='gray') # 使用灰色“喷涂”图像输出显示
plt.show()                   # 显示图像
```



请注意，在前面的图像中，存在的四个片段，显示了在两个图像中仅改变了位置的两个棋子的初始和最终位置。

实验小结

在本实验中，我们使用 XOR 操作执行运动检测，以检测棋盘上移动过其位置的两个棋子。假如您希望进一步了解按位运算（Bitwise Operations）的示例，可以参考官方文档说明 (https://docs.opencv.org/4.2.0/d0/d86/tutorial_py_image_arithmetics.html)。