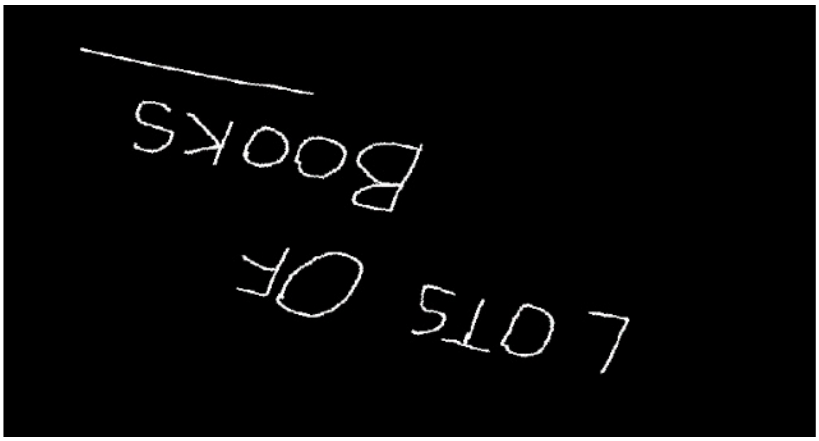
案例：识别影印文档字符

案例目标

扫描您文档时，通常需要保持纸张稳定，从而使文本保持水平的笔直方向。实际上，利用计算机视觉应用程序能够构建一个智能系统，即使该文件被倒转显示，或存在不对称图像，该系统也可以读取该文档。这种应用程序所需的基本工具之一是：即使字符被倒置，翻转，镜像或倾斜也可以识别字符。

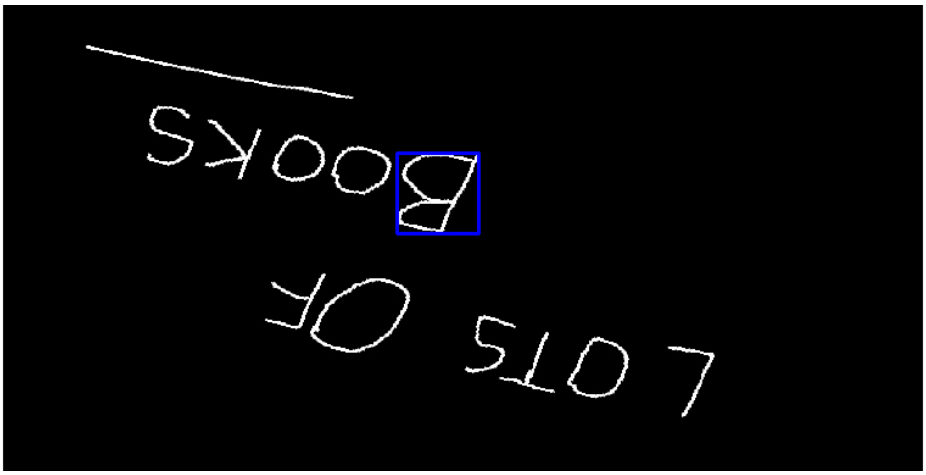
我们会为您提供两张图片，一张是被翻转扫描的手写便笺：



第二张图像（下图）用于让您明确告诉程序：标准的字母 B 长什么样子。 您的任务是编写一个程序，以识别上图中的字母 B 出现在何处。

本案例的任务是创建一个程序，该程序将根据给定提供的参考图像 B ，识别手写便笺中字母 B 的位置。

您的程序应生成以下输出：



案例详细操作说明

- 手写便笺图像路径：`./data/phrase_handwritten.png`
- 参考图像 B 图像路径：`./data/typed_B.png`

1. 导入必要的库并拷贝原始手写便笺图像以备用。
2. 使用 OTSU （大津法）将手写便笺图像（`./data/phrase_handwritten.png`）转换为二值图像 —— 请注意考虑是否需要转换灰度图像与反转图像。
3. 使用 `cv2.RETR_EXTERNAL` 方法在手写便笺图像上找到所有最外层的轮廓，并且使用矩形边框全部绘制出来（可以在原图像上）。
4. 使用相同的流程对参考图像 B （`./data/typed_B.png`）执行 OTSU （大津法）二值化与找到所有最外层的轮廓。
5. 将手写便笺图像上检测到的每个轮廓与参考图像上检测到的轮廓进行比较，通过创建一个新的空列表（譬如：`dist_list`）进行记录轮廓匹配数值差。
6. 可以使用 `min` 命令，在上一步的列表（`dist_list`）中，找到最接近的匹配项。
7. 您可以根据最接近的匹配项的数值差，使用 `dist_list.index` 命令在 `dist_list` 列表中找到对应轮廓的索引号；之后将该索引号代入手写便笺图像的轮廓列表，找到对应的轮廓；最后，在手写便笺的图像上（原图像的拷贝副本），标记与参考图像上的轮廓最相似的轮廓并用矩形边框将它包围起来。

案例小结

在本案例中，我们学习了如何应用轮廓检测来识别字母 B 。 同样的技术也可以用于检测其他字母字符，数字和特殊字符：任何可以表示为单斑点的对象。请注意，参考图像中的 B 并非很正规传统的计算机印刷体，而是采用非常前卫的字体以及复杂的格式书写。您可以从中感受到 OpenCV 轮廓匹配技术的优势。

如果您想进一步探索它，可以使用手写的 B 或其他一些标准格式键入的图像，然后将其用作参考图像，测试是否能在手写便笺图像上成功的将 B 匹配出来？

到目前为止，您了解了轮廓是对象的形状勾画。该轮廓可以在形状的外部边界（外部轮廓）上，也可以在对象内部的孔或其他中空表面上（内部轮廓）。您学习了如何检测不同对象的轮廓，以及如何使用不同的颜色将其绘制在图像上。此外，您还学习了如何根据其轮廓面积，宽度和高度比例等特征来定位不同的轮廓。最后，您完成了在包含多个轮廓的图像中，匹配参考形状的轮廓的实践练习。这可以作为各种智能系统，例如：车牌识别系统的基础技术。

案例答案

In [1]:

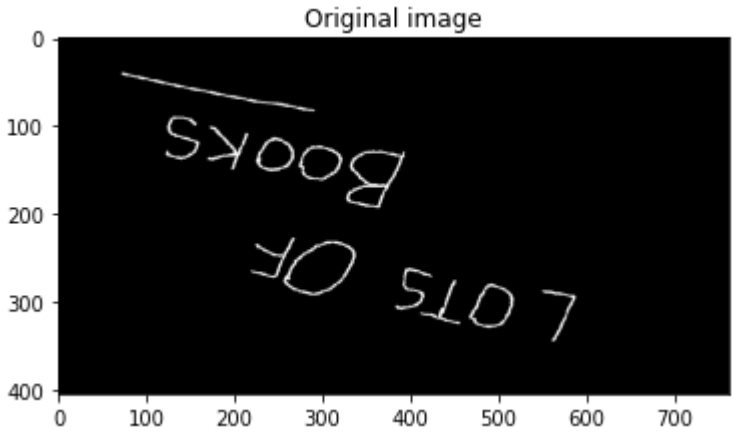
```
# 导入必要的库
import cv2                          # 导入OpenCV
import numpy as np                  # 导入numPy
import matplotlib.pyplot as plt     # 导入matplotlib

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

# 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 + "lab4/")
result_path = "result/"
os.makedirs(result_path, exist_ok=True)

# 读取手写便笺图像
image = cv2.imread('./data/phrase_handwritten.png')
# 读取手写便笺原图像副本备用
imagecopy= image.copy()

# 将图像从 BGR 转换为 RGB
plt.imshow(image[:,:,::-1])
# 指定输出图像的标题
plt.title('Original image')
# 显示图像
plt.show()
```



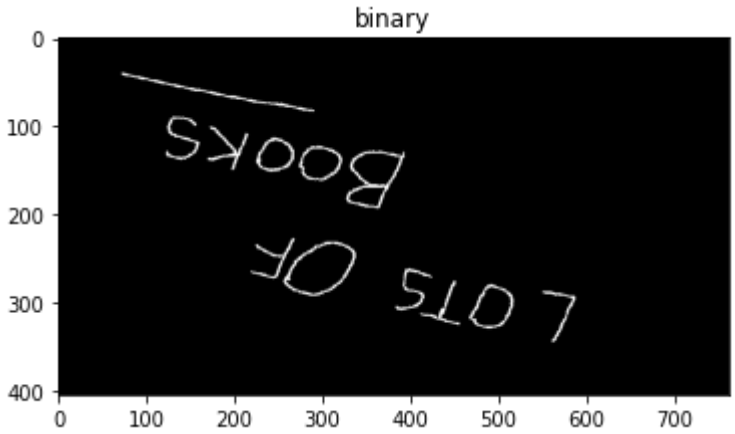
In [2]:

```
# 将图像转换为灰度图像
gray_image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)

# 使用OTSU（大津法）执行图像二值化
ret,binary_im = cv2.threshold(gray_image,0,255,cv2.THRESH_OTSU)

# cv2.imshow( 'binary image' , binary_im )
# cv2.waitKey(0)
# cv2.destroyAllWindows()

# 使用灰色“喷涂”图像输出显示
plt.imshow(binary_im, cmap='gray')
# 指定输出图像的标题
plt.title('binary')
# 显示图像
plt.show()
```



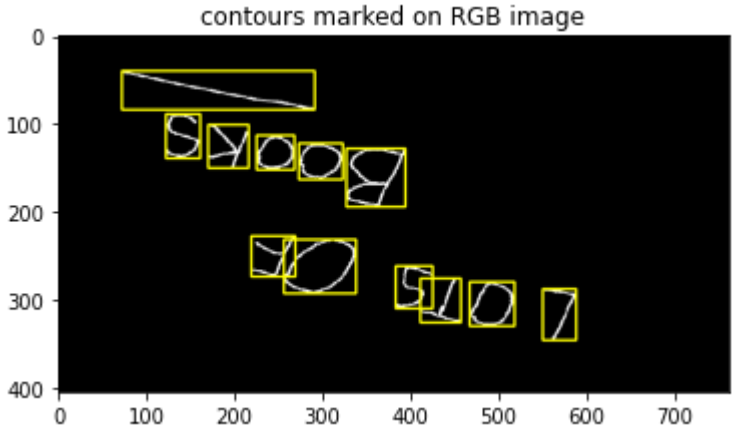
In [3]:

```
# 从二值图像中找到外部轮廓
contours_list,_ = cv2.findContours(binary_im,
                          cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)


# 使用for循环逐个轮询contours_list内的轮廓
for cnt in contours_list:
    # 将轮廓传递到cv2.boundingRect函数中来获取每一个轮廓的x/y/w/y参数
    x,y,w,h = cv2.boundingRect(cnt)
    # 在原始彩色图像image上绘制此边界框
    cv2.rectangle(image,(x,y),(x+w,y+h),(0, 255, 255),2)

# cv2.imshow( 'Contours marked on RGB image' , image )
# cv2.waitKey(0)
# cv2.destroyAllWindows()

# 将图像从 BGR 转换为 RGB
plt.imshow(image[:,:,::-1])
# 指定输出图像的标题
plt.title('contours marked on RGB image')
# 显示图像
plt.show()
```



In [4]:

```
# 将图像转换为灰度图像
ref_gray = cv2.imread('./data/typed_B.png', cv2.IMREAD_GRAYSCALE)

# 使用OTSU（大津法）执行图像二值化
ret, ref_binary = cv2.threshold(ref_gray,0,255,cv2.THRESH_OTSU)

# cv2.imshow( 'Reference image' , ref_binary )
# cv2.waitKey(0)
# cv2.destroyAllWindows()

# 从二值图像中找到外部轮廓
ref_contour_list,_ = cv2.findContours(ref_binary,
                            cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

In [5]:

```
# 首先检查参考图像中是否只有一个轮廓`B`
# 如果轮廓列表中有且只有一个轮廓，条件成立
if len(ref_contour_list)==1:
    # 定义ref_contour为将该轮廓索引的第一个轮廓，用于后面的匹配
    ref_contour= ref_contour_list[0]

# 否则，显示系统警告提示并退出
else:
    import sys
    print('Reference image contains more than 1 contour. Please check!')
    sys.exit()


# ctr= 0
# 创建空列表，用于保存每次轮廓匹配后的数值差
dist_list= []
# 创建contours_list轮廓列表索引号码的for循环
for cnt in contours_list:

    # 按照contours轮廓列表索引号码提取轮廓，与reference_contour参考轮廓进行匹配
    retval = cv2.matchShapes(cnt, ref_contour,cv2.CONTOURS_MATCH_I1,0)
    # 将本次轮廓匹配数值差填入dist_list列表，之后执行下一次循环
    dist_list.append(retval)
#    ctr= ctr+1

print(dist_list) # 输出轮廓匹配数值差列表
```

```
[20.27169325926344, 0.800529118425284, 138.05755161041387, 41.894436731902644, 0.675
1009682610636, 27.888922827660945, 0.5993755151002665, 0.8430159592312115, 0.8525171
556680327, 19.775555289690196, 9.272941532050803, 12.028023300111508]
```

In [6]:

```
min_dist= min(dist_list) # 找到最接近的匹配项

# 可选，输出最接近匹配项的轮廓匹配数值差
print('The minimum distance of the reference contour with a contour in the main image is ' + str(min
```

```
The minimum distance of the reference contour with a contour in the main image is 0.
5993755151002665
```

In [7]:

```
# 在dist_list列表中具有最小数值差的索引号，代入ind_min_dist
ind_min_dist= dist_list.index(min_dist)
# 根据ind_min_dist索引号找到contours_list轮廓列表中的目标轮廓
required_cnt= contours_list[ind_min_dist]

# 将轮廓传递到cv2.boundingRect函数中来获取每一个轮廓的x/y/w/y参数
x,y,w,h = cv2.boundingRect(required_cnt)
# 在之前保存的原始彩色图像的副本上绘制此边界框
cv2.rectangle(imagecopy,(x,y),(x+w,y+h),(255, 0, 0),2)

# cv2.imshow( 'Detected B' , imagecopy )
# cv2.waitKey(0)
# cv2.destroyAllWindows()

# 将图像从 BGR 转换为 RGB
plt.imshow(imagecopy[:,:,::-1])
# 指定输出图像的标题
plt.title('Detected B')
# 显示图像
plt.show()
```