计算机视觉实验一手册

基础操作

1. Jupyter启动：点击开始-展开Anaconda3(64-bit)-点击Anaconda Powershell Prompt-进入以后将以下代码复制并粘贴进去，进入jupyter notebook

```
D:\bc\Scripts\activate.ps1
jupyter notebook
```

2. 读取图像：读取图像是图像处理的第一步,OpenCV提供了 cv2.imread() 函数来加载图像。

```
import cv2 # 读取图像
img = cv2.imread('path_to_image.jpg') # 替换 'path_to_image.jpg' 为你的图片路径
```

3. 显示图像：可以通过 `cv2.imshow()` 实现

```
# 显示图像
cv2.imshow('Image', img)
cv2.waitKey(0) # 等待按键按下
cv2.destroyAllWindows() # 关闭窗口
```

4. 转换图像颜色空间:图像从BGR颜色空间转换到灰度是一个常见的操作，可以通过 `cv2.cvtColor()` 函数完成。

```
# 将图像从BGR转换为灰度
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

5. 保存图像:处理完图像后，你可能想要保存结果。这可以通过 `cv2.imwrite()` 函数实现。

```
# 保存图像
cv2.imwrite('path_to_save_image.jpg', gray_img)
```

实验任务

1. 修改区域颜色

任务分解

1. **读取图像并打印原始像素值**：
   - 使用 OpenCV 读取 `lenacolor.png`，打印左上角第一个像素 (第 0 行，第 0 列) 的 B、G、R 通道值到命令行。
2. **修改图像区域颜色**：
   - 将图像左上角区域分为以下部分并修改颜色：
     - 0-49 行，0-99 列：白色 (255, 255, 255)。
     - 50-99 行，0-99 列：灰色 (128, 128, 128)。
     - 100-149 行，0-99 列：黑色 (0, 0, 0)。
     - 150-199 行，0-99 列：红色 (0, 0, 255)。
   - 保存修改后的图像为 `modified_lenacolor.png`。
3. **打印修改后的像素值**：
   - 打印修改后左上角第一个像素的 B、G、R 值到命令行。

注意事项

- **函数 `cv2.imread(filename, flags)`**：
  - 功能：读取图像文件并返回 NumPy 数组。
  - 参数：
    - `filename`：图像文件路径（如 "lenacolor.png"）。
    - `flags`：读取模式，默认 `cv2.IMREAD_COLOR`（BGR 格式），可选 `cv2.IMREAD_UNCHANGED`（保留原始格式）。
  - 注意：返回的图像为 BGR 格式，通道顺序为 [B, G, R]，若文件不存在返回 `None`。
- 使用 `img[row, col, channel]` 访问具体颜色通道，例如 `img[0, 0, 0]` 为蓝色通道值。
- 使用 `img[y1:y2, x1:x2] = [B, G, R]` 修改区域颜色，注意范围不包含右边界（如 `0:50` 是 0-49）。
- **函数 `cv2.imwrite(filename, img)`**：
  - 功能：将图像保存到指定文件。

    - 参数：
        - `filename`：保存路径（如 "modified_lenacolor.png"）。
        - `img`：要保存的图像数组。
    - 注意：确保路径正确，否则保存失败。
- 使用 `print()` 输出像素值到命令行，格式化字符串便于阅读。

结果参考

---

2. 读取彩色图片并创建掩码

任务分解

1. **读取图像并转换**：
    - 读取彩色图片 `x.jpg`，将其从 BGR 转换为灰度图像。
    - 创建一个掩码 (mask) 并保存为 `mask.png`。
    - 将灰度图像保存为 `gray_x.jpg`。

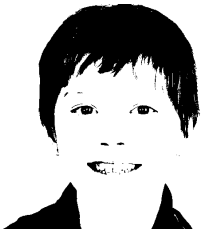
注意事项

- **函数 `cv2.cvtColor(src, code)`**：
    - 功能：转换图像颜色空间。
    - 参数：
        - `src`：输入图像。
        - `code`：转换类型，如 `cv2.COLOR_BGR2GRAY`（BGR 转灰度）。
    - 注意：灰度图像为单通道，值范围 0-255。
- **函数 `cv2.threshold(src, thresh, maxval, type)`**：
    - 功能：对图像进行阈值处理，生成二值掩码。
    - 参数：

- `src`：输入图像（灰度）。
- `thresh`：阈值（如 127）。
- `maxval`：大于阈值时设定的值（如 255）。
- `type`：阈值类型，如 `cv2.THRESH_BINARY`（大于阈值为 maxval，小于为 0）。

- 注意：返回值为 `(ret, dst)`，`ret` 为阈值，`dst` 为结果图像。

结果参考

---

3. 形态学操作（基于图像 `"4.jpg"`、`"5.jpg"` 和 `"6.jpg"`）

任务分解

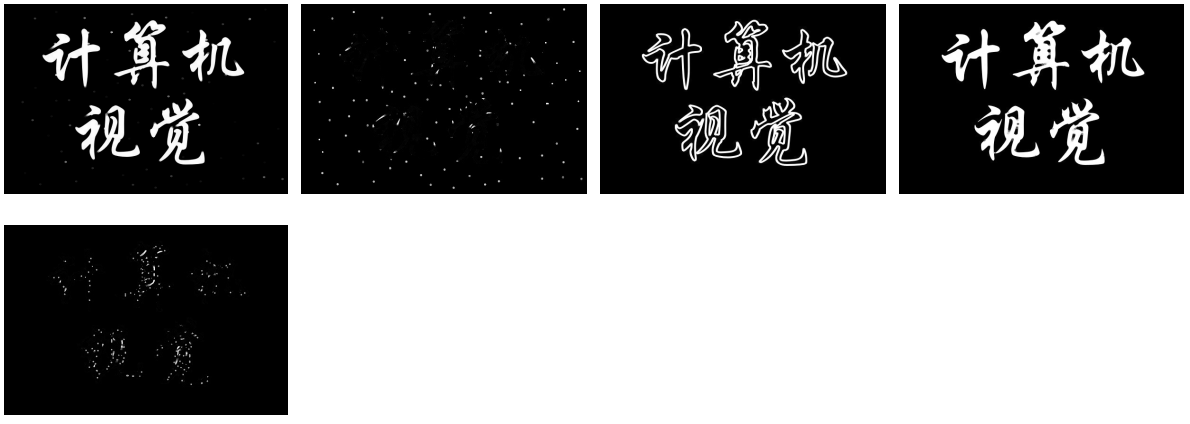
1. **处理图像 `"4.jpg"`**：
   - 将图像转换为灰度。
   - 使用 5x5 椭圆形结构元素执行开运算，保存结果为 `open_4.jpg`。
   - 使用 5x5 椭圆形结构元素执行顶帽操作，保存结果为 `tophat_4.jpg`。
2. **处理图像 `"5.jpg"`**：
   - 将图像转换为灰度。
   - 使用 5x5 椭圆形结构元素执行形态学梯度操作，保存结果为 `gradient_5.jpg`。
3. **处理图像 `"6.jpg"`**：
   - 将图像转换为灰度。
   - 使用 5x5 椭圆形结构元素执行闭运算，保存结果为 `close_6.jpg`。
   - 使用 5x5 椭圆形结构元素执行黑帽操作，保存结果为 `blackhat_6.jpg`。

注意事项

- **结构元素**：
  - **函数 `cv2.getStructuringElement(shape, ksize)`**：

    - 功能：生成指定形状和大小的结构元素。
    - 参数：
        - `shape`：形状类型，如 `cv2.MORPH_ELLIPSE`（椭圆形）。
        - `ksize`：结构元素大小，如 `(5, 5)`。
    - 注意：5x5 椭圆形核用于所有操作，需保持一致。
- **形态学操作**：
    - **函数 `cv2.morphologyEx(src, op, kernel)`**：
        - 功能：执行高级形态学操作（如开运算、闭运算等）。
        - 参数：
            - `src`：输入图像（灰度）。
            - `op`：操作类型，如 `cv2.MORPH_OPEN`（开运算）、`cv2.MORPH_CLOSE`（闭运算）、`cv2.MORPH_TOPHAT`（顶帽）、`cv2.MORPH_BLACKHAT`（黑帽）、`cv2.MORPH_GRADIENT`（梯度）。
            - `kernel`：结构元素。
        - 注意：
            - 开运算（`MORPH_OPEN`）：先腐蚀后膨胀，去除小亮点。
            - 闭运算（`MORPH_CLOSE`）：先膨胀后腐蚀，填补小空洞。
            - 顶帽（`MORPH_TOPHAT`）：原图与开运算结果之差，突出亮细节。
            - 黑帽（`MORPH_BLACKHAT`）：闭运算结果与原图之差，突出暗细节。
            - 梯度（`MORPH_GRADIENT`）：膨胀与腐蚀结果之差，提取边缘。

结果参考



4. 图像透视变换与 OCR 文字识别

任务分解

1. **图像预处理**：
   - 读取图像并调整大小，保持宽高比。
   - 转换为灰度图，应用高斯模糊去噪。
   - 使用 Canny 边缘检测提取边缘。
2. **轮廓检测**：
   - 检测图像中的轮廓，按面积排序并筛选四边形轮廓。
   - 在图像上绘制绿色轮廓以标记目标区域。
3. **透视变换**：
   - 对检测到的四个顶点进行排序。
   - 计算目标区域的宽度和高度，应用透视变换生成扫描效果。
   - 对变换后的图像进行二值化处理。
4. **OCR 文字识别**：
   - 对扫描图像进行预处理（可选阈值化或模糊）。
   - 使用 Tesseract OCR 提取图像中的文字。
   - 保存处理结果并打印识别文字。
5. **结果保存**：
   - 保存每个步骤的中间结果和最终扫描及 OCR 处理图像。

注意事项

- **函数 `cv2.resize(image, dsize, interpolation)`**：
  - 功能：调整图像大小。
  - 参数：
    - `image`：输入图像。
    - `dsize`：目标尺寸（如 (width, height)）。
    - `interpolation`：插值方法（如 `cv2.INTER_AREA`）。
  - 注意：保持宽高比以避免失真。

- **函数 `cv2.Canny(image, threshold1, threshold2)`**：
  - 功能：边缘检测。
  - 参数：
    - `image`：输入灰度图像。
    - `threshold1`：低阈值（如 75）。
    - `threshold2`：高阈值（如 200）。
  - 注意：阈值需根据图像调整以平衡边缘完整性和噪声。
- **函数 `cv2.getPerspectiveTransform(src, dst)` 和 `cv2.warpPerspective(image, M, dsize)`**：
  - 功能：计算并应用透视变换。
  - 参数：
    - `src`：源坐标点。
    - `dst`：目标坐标点。
    - `image`：输入图像。
    - `M`：变换矩阵。
    - `dsize`：输出尺寸。
  - 注意：坐标点需正确排序以避免畸变。
- **函数 `pytesseract.image_to_string(image, config)`**：
  - 功能：OCR 文字提取。
  - 参数：
    - `image`：输入图像（PIL 格式）。
    - `config`：ocr库配置路径
  - 注意：需安装 Tesseract 并配置环境变量或指定 `tessdata` 路径。

---

5. 多人脸检测

任务分解

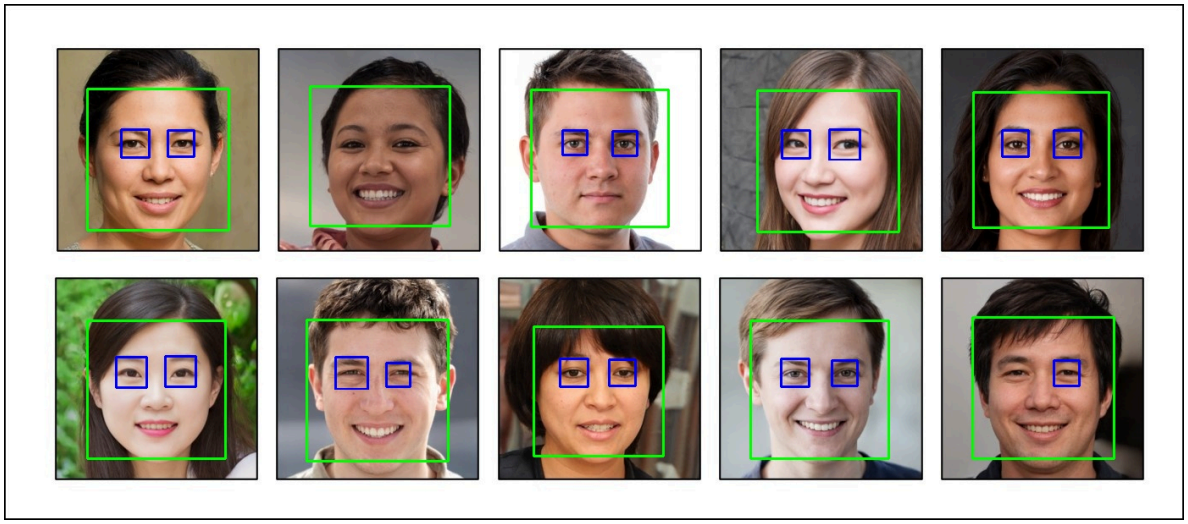
1. **读取图像并预处理**：
   - 读取彩色图像并转换为灰度图像，为后续检测做准备。

2. **人脸检测**：
    - 使用 Haar 级联分类器检测图像中的人脸。
    - 在检测到的人脸上绘制绿色矩形框。
3. **眼睛检测**：
    - 在检测到的人脸区域（上半部分）内进行眼睛检测。
    - 在检测到的眼睛上绘制蓝色矩形框。
4. **结果保存**：
    - 将标记了人脸和眼睛的图像保存为新文件。

注意事项

- **函数 `cv2.CascadeClassifier.detectMultiScale(image, scaleFactor, minNeighbors, minSize)`**：
    - 功能：检测多尺度目标（人脸或眼睛）。
    - 参数：
        - `image`：输入图像（灰度）。
        - `scaleFactor`：图像缩放比例（如 1.04 或 1.05），影响检测速度和精度。
        - `minNeighbors`：邻域矩形数量阈值（如 18 或 10），值越高误检越少但可能漏检。
        - `minSize`：目标最小尺寸（如 (8, 8) 或 (22, 22)），需根据实际情况调整。
    - 注意：参数需根据图像分辨率和目标大小优化。
- **函数 `cv2.rectangle(image, pt1, pt2, color, thickness)`**：
    - 功能：在图像上绘制矩形。
    - 参数：
        - `image`：目标图像。
        - `pt1`，`pt2`：矩形左上角和右下角坐标。
        - `color`：矩形颜色（如 (0, 255, 0) 表示绿色）。
        - `thickness`：线条粗细（如 2）。
    - 注意：坐标需在图像范围内。

结果参考



---

6. 基于 MeanShift 的目标跟踪

任务分解

1. **读取视频并设置初始跟踪区域**：
   - 读取视频文件 `video.mp4`，设置初始跟踪窗口（例如 `[800, 400, 200, 160]`）。
2. **计算目标直方图并跟踪**：
   - 将初始区域转换为 HSV 颜色空间，计算色调（H）通道的直方图。
   - 使用 MeanShift 算法跟踪目标，更新窗口位置。
3. **保存跟踪结果**：
   - 将跟踪窗口绘制在每帧上，保存为视频文件 `tracked_video.mp4`。

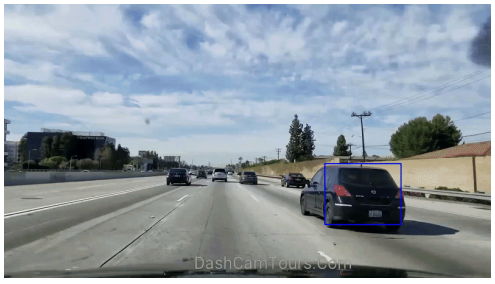
注意事项

- **函数 `cv2.calcHist(images, channels, mask, histSize, ranges)`**：
  - 功能：计算图像直方图。
  - 参数：
    - `images`：输入图像列表（如 `[hsv_roi]`）。
    - `channels`：计算通道（如 `[0]` 表示 H 通道）。
    - `mask`：掩码，限制计算区域。
    - `histSize`：直方图 bin 数（如 `[180]`）。
    - `ranges`：值范围（如 `[0, 180]`）。

- 注意：用于描述目标颜色分布，需归一化后使用。

- **函数 `cv2.meanShift(probImage, window, criteria)`**：
  - 功能：执行 MeanShift 跟踪，返回新窗口位置。
  - 参数：
    - `probImage`：反投影图像。
    - `window`：初始窗口（x, y, w, h）。
    - `criteria`：终止条件，如 `(cv.TERM_CRITERIA_EPS | cv.TERM_CRITERIA_COUNT, 10, 1)`。
  - 注意：终止条件控制迭代次数和精度，需合理设置。
- **函数 `cv2.VideoWriter(filename, fourcc, fps, frameSize)`**：
  - 功能：创建视频写入对象。
  - 参数：
    - `filename`：输出视频路径。
    - `fourcc`：编码格式，如 `cv2.VideoWriter_fourcc(*'mp4v')`。
    - `fps`：帧率（如 30.0）。
    - `frameSize`：帧尺寸（如 `(width, height)`）。
  - 注意：确保帧尺寸与输入一致，否则保存失败。

结果参考



---

7. 改进任务6

问题：在视频后段，车辆的跟踪出现了漂移，效果变得不好了
目标：修改代码，提高视频后段的跟踪效果
提示：当前帧中直方图反应大部分区域发生变化时，剩下不变的区域就会被错误跟踪

---

8. 车道线检测

任务分解

1. **图像预处理**：
   - 将输入图像转换为灰度图。
   - 应用高斯滤波去噪。
   - 使用 Canny 算法进行边缘检测。
2. **感兴趣区域提取**：
   - 定义并应用掩模，保留图像中车道线所在的区域。
3. **车道线检测**：
   - 使用霍夫变换检测直线。
   - 根据斜率和位置筛选并分离左右车道线。
   - 绘制左右车道线。
4. **结果融合与保存**：
   - 将检测到的车道线与原图像融合。
   - 保存处理后的图像。

注意事项

- **函数 `cv2.Canny(image, threshold1, threshold2)`**：
  - 功能：执行 Canny 边缘检测。
  - 参数：
    - `image`：输入灰度图像。
    - `threshold1`：低阈值（如 75），用于边缘连接。
    - `threshold2`：高阈值（如 225），用于强边缘检测。
  - 注意：阈值需根据图像调整，过低噪声多，过高边缘缺失。
- **函数 `cv2.GaussianBlur(image, ksize, sigma)`**：
  - 功能：高斯滤波去噪。
  - 参数：
    - `image`：输入图像。
    - `ksize`：核大小（如 (5, 5)），必须为奇数。

    - `sigma`：标准差（如 0 表示自动计算）。
  - 注意：核大小影响平滑程度，过大会丢失细节。
- **函数 `cv2.HoughLinesP(image, rho, theta, threshold, minLineLength, maxLineGap)`**：
  - 功能：概率霍夫变换检测直线。
  - 参数：
    - `rho`：距离精度（如 1 像素）。
    - `theta`：角度精度（如 π/180 弧度）。
    - `threshold`：累加阈值（如 20）。
    - `minLineLength`：最小线段长度（如 30 像素）。
    - `maxLineGap`：最大断裂长度（如 60 像素）。
  - 注意：参数需根据图像分辨率和车道线特征优化。
- **函数 `cv2.addWeighted(src1, alpha, src2, beta, gamma)`**：
  - 功能：图像加权融合。
  - 参数：
    - `src1`, `src2`：输入图像。
    - `alpha`, `beta`：两图像权重（如 0.8 和 1.0）。
    - `gamma`：亮度调整（如 0）。
  - 注意：权重和决定融合效果，需保持总和合理。
- **斜率筛选**：
  - 使用 `slope_min`（如 0.35）和 `slope_max`（如 0.85）过滤非车道线。
  - 注意：范围需根据实际车道线角度调整。

结果参考

附加实验1: SIFT 特征匹配与图像拼接

任务分解

1. **读取图像并调整尺寸**:
   - 读取 `"image1.jpg"`，`"image2.jpg"和"image3.jpg"`，调整为 640x480。
2. **SIFT 特征检测与匹配**:
   - 检测两张图像的 SIFT 关键点，保存带关键点的图像为 `sift_keypoints_image1.jpg` 和 `sift_keypoints_image2.jpg` 。
   - 使用 BFMatcher 进行 k-NN 匹配，绘制并保存所有匹配关系为 `sift_matches.jpg` ，筛选优质匹配后保存为 `sift_good_matches.jpg` 。
3. **计算单应性矩阵并拼接**:
   - 使用 RANSAC 计算单应性矩阵，拼接图像，保存结果为 `stitched_image.jpg` 。

注意事项

- `cv.SIFT_create()` ： 创建 SIFT 检测器。
- `sift.detectAndCompute(image, None)` ： 检测关键点和描述符。
- `cv.drawKeypoints()` ： 绘制关键点。
- `cv.BFMatcher()` 和 `bf.knnMatch()` ： 暴力匹配和 k-NN 匹配。
- `cv.drawMatches()` ： 绘制匹配关系。

- `cv.findHomography()`：计算单应性矩阵。
- `cv.warpPerspective()` 和 `cv.perspectiveTransform()`：图像变换与拼接。

结果参考

附加实验2: 细胞轮廓识别与编号标注

任务分解

1. **图像预处理**：
   - 读取彩色图像并转换为灰度图像。
   - 应用反转二值化处理，分离目标与背景。
   - 使用形态学腐蚀去除噪声，随后膨胀恢复目标形状。
   - 应用高斯模糊平滑图像。
2. **轮廓检测**：
   - 检测图像中的轮廓并筛选面积大于 30 的有效轮廓。
3. **轮廓标注**：
   - 在原图像上绘制绿色轮廓。
   - 计算每个轮廓的质心并标注红色编号。
4. **结果保存**：

- 保存高斯模糊后的图像。
- 保存绘制了轮廓和编号的最终图像。

注意事项

- **函数 `cv2.threshold(image, thresh, maxval, type)`**：
  - 功能：图像二值化。
  - 参数：
    - `image`：输入灰度图像。
    - `thresh`：阈值（如 150）。
    - `maxval`：最大值（如 255）。
    - `type`：类型（如 `cv2.THRESH_BINARY_INV` 表示反转二值化）。
  - 注意：阈值需根据图像亮度调整，过低或过高会影响目标分离。
- **函数 `cv2.erode(image, kernel, iterations)`** 和 **`cv2.dilate(image, kernel, iterations)`**：
  - 功能：形态学腐蚀和膨胀。
  - 参数：
    - `image`：输入图像。
    - `kernel`：结构元素（如 5x5 椭圆核）。
    - `iterations`：迭代次数（如 4 或 3）。
  - 注意：迭代次数影响处理强度，需平衡噪声去除和目标保留。
- **函数 `cv2.findContours(image, mode, method)`**：
  - 功能：检测轮廓。
  - 参数：
    - `image`：输入二值图像。
    - `mode`：检索模式（如 `cv2.RETR_TREE` 表示层次结构）。
    - `method`：轮廓近似方法（如 `cv2.CHAIN_APPROX_SIMPLE` 简化点）。
  - 注意：输入图像需为二值化图像。
- **函数 `cv2.drawContours(image, contours, contourIdx, color, thickness)`**：
  - 功能：绘制轮廓。

- 参数：
  - `image`：目标图像。
  - `contours`：轮廓列表。
  - `contourIdx`：绘制索引（-1 表示所有轮廓）。
  - `color`：颜色（如 (0, 255, 0) 为绿色）。
  - `thickness`：线条粗细（如 1）。
- 注意：图像需为彩色格式。

- **函数 `cv2.putText(image, text, org, font, fontScale, color, thickness)`**：
  - 功能：在图像上添加文本。
  - 参数：
    - `image`：目标图像。
    - `text`：文本内容（如轮廓编号）。
    - `org`：文本位置（如质心坐标）。
    - `font`：字体（如 `cv2.FONT_HERSHEY_PLAIN`）。
    - `fontScale`：字体大小（如 1.5）。
    - `color`：颜色（如 (0, 0, 255) 为红色）。
    - `thickness`：粗细（如 2）。
  - 注意：坐标需在图像范围内。

结果参考