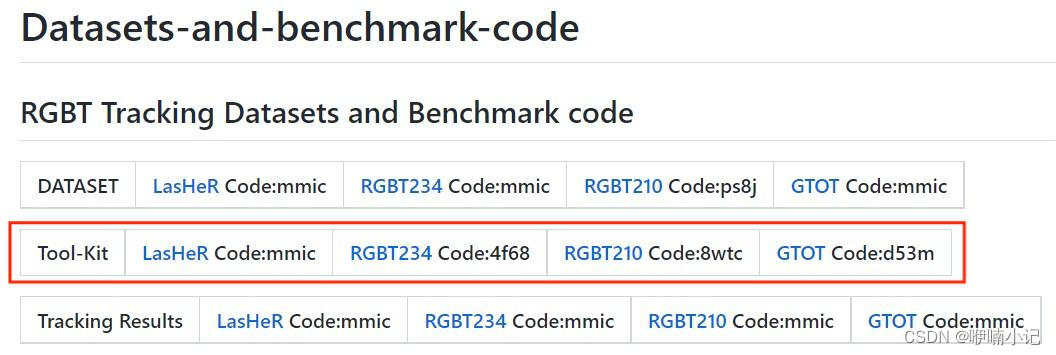
测评工具箱使用

参考资料：[1] [GTOT和RGBT234测评工具箱使用](https://blog.csdn.net/m0_46749624/article/details/127668031?spm=1001.2014.3001.5501)

**0 环境**

* 下载测评工具箱，链接：[GitHub - mmic-lcl/Datasets-and-benchmark-code](https://github.com/mmic-lcl/Datasets-and-benchmark-code)



注意：用什么数据集测试，用什么测评工具箱：

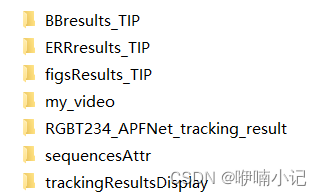
PlotErr234-clear 用RGBT234测试

plot-50-clear 用GTOT测试

1 RGBT234测评工具箱（PlotErr234-clear）使用

**结构**

在RGBT234测评工具箱中，共有七个文件夹，其中， BBresults\_TIP 文件夹中存放的是测试生成的目标跟踪框的 txt 文件，ERRresults\_TIP 文件夹中的是计算生成的错误率文件，figsResults\_TIP文件夹中的是不同属性下的PR、SR曲线评测图，sequencesAttr 是工具箱中自带的关于 RGBT234 数据集特性的一些文件，trackingResultsDisplay是存放带有跟踪框的结果图像的文件夹，RGBT234\_APFNet\_tracking\_result 文件夹存放的是使用GTOT数据集作为训练集， RGBT234作为测试集得到的测试跟踪结果（文件中每一行都含有八个数值，前四个和后四个分别代表在 RGB 图像和热红外图上的跟踪框四元数），my\_video文件夹是自己建立的，用来存放合成视频的文件夹。

·

**准备工作**

在测评工具箱根目录下**新建脚本cp\_BBres\_txts.py**，用于将测试APF模型时生成的bbox框txt文件拷贝并重命名至 BBresults\_TIP 与 ERRresults\_TIP/APF 两个文件夹内，***需要注意的是RGBT234数据集解压后可能自带有一个rgbt234.txt文件，需要将其删除。***

import os

input\_path = './RGBT234\_APFNet\_tracking\_result/'

output\_path = './BBresults\_TIP/'

#output\_path = './ERRresults\_TIP/APF/'

for origin\_txt in os.listdir(input\_path):

os.system('cp ' + input\_path + origin\_txt + ' ' + output\_path + 'APF\_' + origin\_txt)

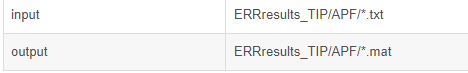
修改output\_path两次，执行脚本两次

python move\_document.py

**测评过程**

**生成ERRresults\_TIP/xxx.mat**

使用main\_GenerateMat\_TPR.m和calcPlotErr\_TPR.m两个脚本，其中main\_GenerateMat\_TPR.m 脚本调用了calcPlotErr\_TPR.m脚本，即main\_GenerateMat\_TPR.m脚本是主程序。该步骤的输入输出分别为：



针对每一个种类的txt文件，都会生成一个mat图表文件；

脚本中有一些参数需要修改：

（1）**main\_GenerateMat\_TPR.m**脚本中：basePath为RGBT234数据集的位置，即"xxx/data/RGBT234/"；trackers为几个追踪算法的名字，这里设置为trackers={'APF'};

（2）**calcPlotErr\_TPR.m**脚本中：basePath为RGBT234数据集的位置；resultPath为跟踪框txt文件所在位置以及要输出的检测框mat文件所在位置，即resultPath='ERRresults\_TIP/APF/'; 需要提前建好APF文件夹;73行左右处也要改为**'APF\_GTOT\_ALL\_Final\_'**，即：

**results.res=dlmread([resultPath 'APF\_GTOT\_ALL\_Final\_' seq\_name '.txt']); %seq\_length\*8**

**这是matlab文件，请在Matlab软件中运行main\_GenerateMat\_TPR.m文件；**

**生成PR、SR曲线图**

使用main\_drawResult\_TPR.m脚本。脚本中有一些参数需要修改：algs为ERRresults\_TIP文件夹下存有的算法的名字；basePath为RGBT234数据集的位置。

%algs={'APF'}; % ours

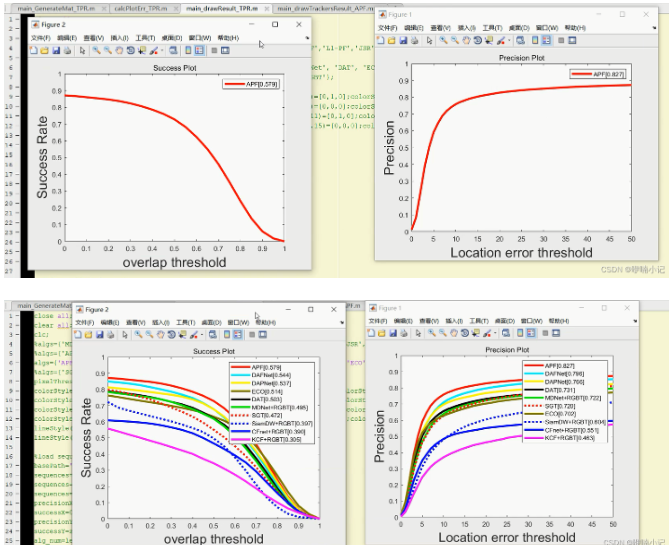
algs={'APF', 'CFnet+RGBT', 'MDNet+RGBT', 'DAFNet', 'KCF+RGBT', 'DAPNet', 'DAT', 'ECO', 'SGT', 'SiamDW+RGBT'}; % overall

pixelThreshold=20;

basePath='xxx/dataset/RGBT234/';

* 该步骤的输入输出为：





* 如上图所示，在PR图中，横轴表示定位损失阈值，单位是像素，纵轴是精确率 PR，论文中说到过，对于RGBT234数据集，阈值设置为20个像素，所以横轴等于20像素时，得到APFNet 在RGBT234数据集上的精确率为 82.7%。各个 RGBT 目标跟踪算法相互对比，可知 APFNet算法在众多算法当中，性能较为优越。
* 在SR图中，横轴表示 overlap 阈值，即追踪框和 GT 框之间交并比的大小，纵轴表示成功率 SR。当阈值取 0.7 时，即当 overlap 大于 0.7 时表示成功追踪，APFNet 算法的 SR 为 57.9%。各个 RGBT 目标跟踪算法相互对比，可知 APFNet 算法在众多算法当中，性能较为优越。

**合成跟踪可视化视频**

使用main\_drawTrackersResult\_APF.m脚本生成追踪结果图片。该脚本复制于main\_drawTrackersResult\_PaGLe.m。其中，tracker为追踪算法的名字，basePath为RGBT234数据集的位置；sequencePath为RGBT234数据集的位置；trackerResultsPath为追踪结果框txt文件所在路径；saveBasePath为连续帧图片保存路径；大约28行的ni为选择某一个序列的序号，也可以直接对sequence赋值决定某一个视频序列。同时需要注意的是展示出来的图片未全部填充整个窗口，所以需要修改大约80和104行的两处imshow代码使图片完全填充窗口，消除白边。

basePath='xxx/dataset/RGB\_T234/';

trackerResultsPath='BBresults\_TIP/';

sequencePath='xxx/dataset/RGB\_T234/';

saveBasePath='trackingResultsDisplay/';

if(isdir(saveBasePath)==0),

mkdir(saveBasePath);

end

tracker={'APF'};

%tracker={'PaGLe','CSR','DSST','HCF','JSR','KCF','L1-PF','MDNet','SGT','SOWP','Staple'};%,'STRUCK','CN','CT','MIL','SCM'};

%all videos, call self with each video name.

%only keep valid directory names

dirs = dir(basePath);

sequences = {dirs.name};

sequences(strcmp('.', sequences) | strcmp('..', sequences) | strcmp('anno', sequences) | ~[dirs.isdir]) = [];

edgeColor={'r','g','b','y','k','m','c','g','b','y','r'};

lineStyle={'-','-','-','-','--','--','--','--','--','--','--'};

% for ni=100:numel(sequences)

for ni=120:120 % 选择第120个文件夹，即manwithluggage

% sequence='walkingtogether1';

sequence=sequences{ni};

%imshow(uint8(im));

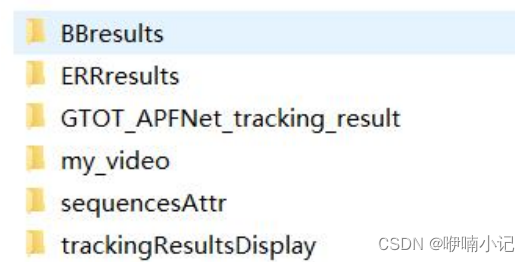
imshow(uint8(im),'border','tight');

* 更换sequence为RGBT234数据集中不同的序列名称，可在trackingResultsDisplay/文件夹下得到不同视频序列在两个模态下的APFNet的跟踪框的可视化图片。
* 在项目根目录下新建my\_video文件夹和gen\_video.py脚本，gen\_video.py用于将trackingResultsDisplay/文件夹下已有的所有序列合成视频。
* 脚本中注意的是上一步保存的图片大小与原图大小不一致，所以脚本中有缩放操作，运行gen\_video.py脚本之后，将在my\_video下得到对应序列名称的所有视频，其中白色框为gt框，红色框为APFNet算法的跟踪框。

**2 GTOT测评工具箱使用**

**结构**

在GTOT 工具箱中，共有六个文件夹，其中， BBresults 文件夹中存放的是测试生成的目标跟踪框的 txt 文件，ERRresults 文件夹中的是计算生成的错误率文件，sequencesAttr 是工具箱中自带的关于 GTOT 数据集特性的一些文件，trackingResultsDisplay是存放带有跟踪框的结果图像的文件夹，GTOT\_APFNet\_tracking\_result 文件夹存放的是使用RGBT234 数据集作为训练集，GTOT 作为测试集得到的测试跟踪结果（文件中每一行都含有八个数值，前四个和后四个分别代表在 RGB 图像和热红外图上的跟踪框四元数），my\_video文件夹是自己建立的，用来存放合成视频的文件夹。



**准备工作**

        在测评工具箱根目录下新建脚本cp\_BBres\_txts.py，用于将测试APF模型时生成的bbox框文件拷贝并重命名至 BBresults 与 ERRresults/APF 两个文件夹内

import os

input\_path = './GTOT\_APFNet\_tracking\_result/'

#output\_path = './BBresults/'

output\_path = './ERRresults/APF/'

for origin\_txt in os.listdir(input\_path):

os.system('cp ' + input\_path + origin\_txt + ' ' + output\_path + 'APF\_' + origin\_txt)

修改output\_path两次，执行脚本两次

python cp\_BBres\_txts.py

python ./plot-50-clear/plot-50/cp\_BBres\_txts.py

测评过程

**生成ERRresults/xxx.mat**

* 使用main\_GenerateMat.m和calcPlotErr.m两个脚本，其中main\_GenerateMat.m 脚本调用了 calcPlotErr.m 脚本，即main\_GenerateMat.m 脚本是主程序。该步骤的输入输出分别为：



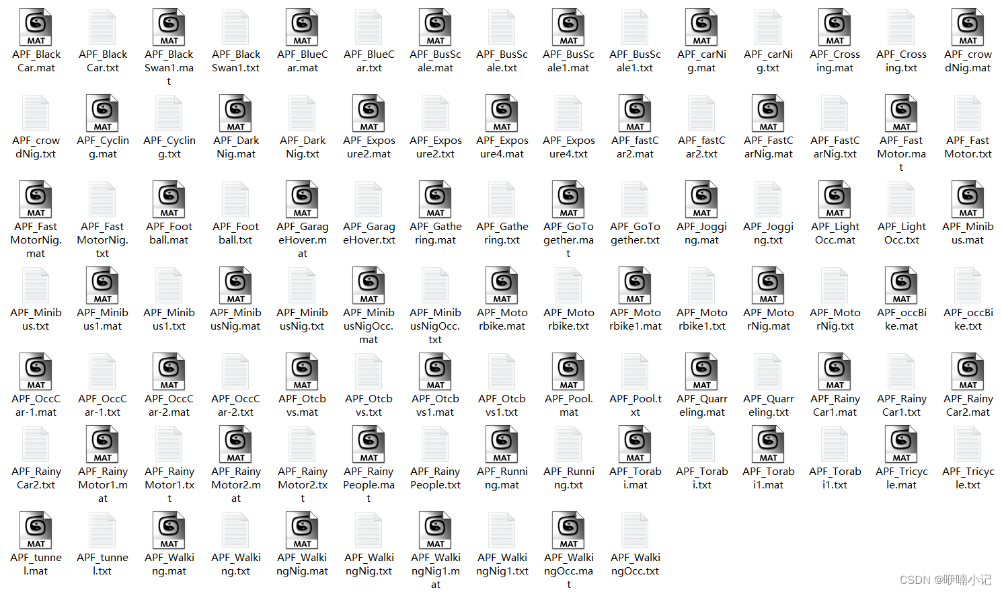
脚本中有一些参数需要修改：

（1）main\_GenerateMat.m脚本中：basePath为GTOT数据集的位置，即"xxx/data/GTOT/"；trackers为几个追踪算法的名字，这里设置为trackers={'APF'};

（2）calcPlotErr.m脚本中：basePath为GTOT数据集的位置；resultPath为跟踪框txt文件所在位置以及要输出的检测框mat文件所在位置，即resultPath='ERRresults/APF/'; 64行左右处也要改为'APF\_'，即：

results.res=dlmread([resultPath 'APF\_' seq\_name '.txt']); %seq\_length\*8

* 运行main\_GenerateMat.m 脚本之后，将在ERRresults/APF下得到相应的mat文件



**生成PR、SR**[**曲线图**](https://so.csdn.net/so/search?q=%E6%9B%B2%E7%BA%BF%E5%9B%BE&spm=1001.2101.3001.7020)

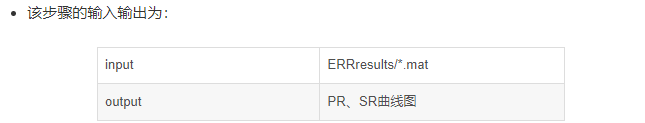
* 使用main\_drawResult.m脚本。脚本中有一些参数需要修改：algs为ERRresults文件夹下存有的算法的名字；basePath为GTOT数据集的位置。

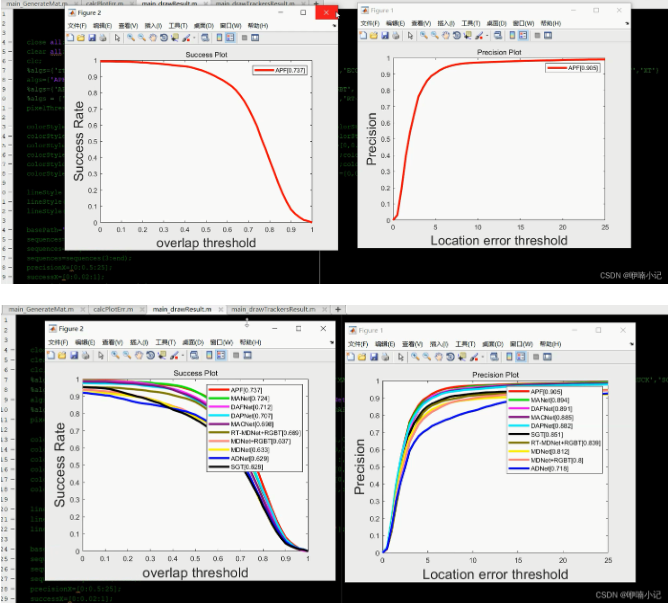
%algs={'APF'}; % ours

algs={'APF', 'ADNet', 'MANet', 'DAPNet', 'DAFNet', 'MDNet', 'SGT', 'RT-MDNet+RGBT', 'MDNet+RGBT', 'MACNet'}; % overall

pixelThreshold=5;

basePath='xxx/dataset/GTOT/';





* 如上图所示，在PR图中，横轴表示定位损失阈值，单位是像素，纵轴是精确率 PR，论文中说到过，对于 GTOT 数据集，阈值设置为 5 个像素，所以横轴等于 5 像素时，得到APFNet 在GTOT数据集上的精确率为 90.5%。各个 RGBT 目标跟踪算法相互对比，可知 APFNet算法在众多算法当中，性能较为优越。
* 在SR图中，横轴表示 overlap 阈值，即追踪框和 GT 框之间交并比的大小，纵轴表示成功率 SR。当阈值取 0.7 时，即当 overlap 大于 0.7 时表示成功追踪，APFNet 算法的 SR 为 73.7%。各个 RGBT 目标跟踪算法相互对比，可知 APFNet 算法在众多算法当中，性能较为优越。

**合成跟踪可视化视频**

使用main\_drawTrackersResult.m脚本生成追踪结果图片。tracker为追踪算法的名字，basePath为GTOT数据集的位置；sequence为视频序列的名字；sequencePath为GTOT数据集的位置；同时需要注意的是展示出来的图片未全部填充整个窗口，所以需要修改大约80和104行的两处imshow代码使图片完全填充窗口，消除白边。

tracker={'APF'} %使用几种跟踪算法就会同时有几种追踪框

basePath='xxx/dataset/GTOT/';

dirs = dir(basePath);

sequences = {dirs.name};

sequences(strcmp('.', sequences) | strcmp('..', sequences) | strcmp('anno', sequences) | ~[dirs.isdir]) = [];

%sequence='Minibus1';

sequence='Football';

%for ni=8

% sequence=sequences{ni}

trackerResultsPath='BBresults/';

sequencePath='xxx/dataset/GTOT/';

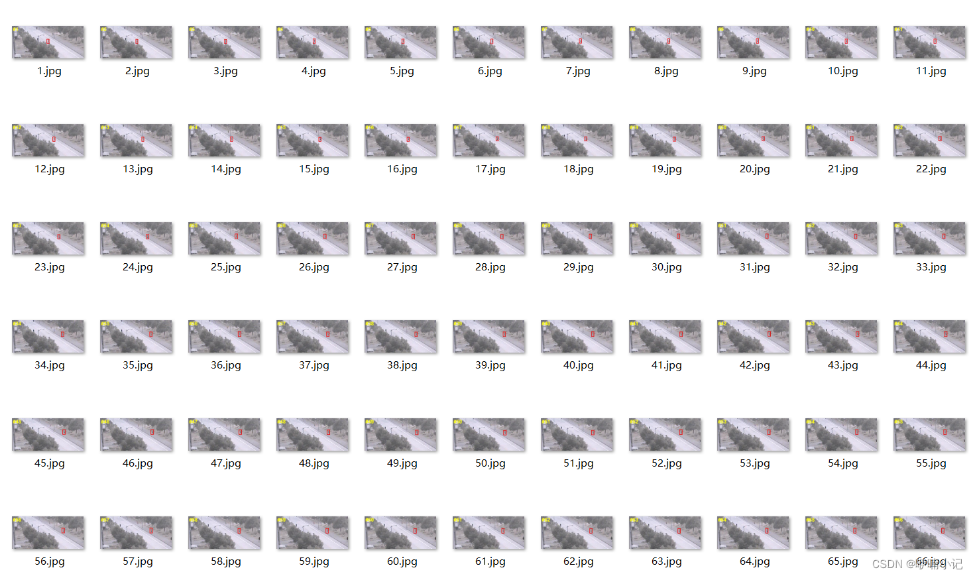
saveBasePath='trackingResultsDisplay/';

-------

%imshow(uint8(im));

imshow(uint8(im),'border','tight');

* 更换sequence为GTOT数据集中不同的序列名称，可在trackingResultsDisplay/文件夹下得到不同视频序列在两个模态下的APFNet的跟踪框的可视化图片。



* 在项目根目录下新建my\_video文件夹和gen\_video.py脚本，gen\_video.py用于将trackingResultsDisplay/文件夹下已有的所有序列合成视频，gen\_video.py内容如下：

import cv2

import os

import linecache

DEBUG = 1

def gen\_video(sequence\_path, res\_video\_path, gt\_path):

if not os.path.exists(res\_video\_path):

os.system('mkdir ' + res\_video\_path)

f = cv2.VideoWriter\_fourcc('X', 'V', 'I', 'D') # 保存视频格式为avi

for sequence\_name in os.listdir(sequence\_path):

#save\_sequence\_path = os.path.join(res\_video\_path, sequence\_name)

save\_sequence\_path = res\_video\_path + '/' + sequence\_name

if DEBUG:

print(save\_sequence\_path)

if not os.path.exists(save\_sequence\_path):

os.makedirs(save\_sequence\_path)

test\_frame\_v = cv2.imread(os.path.join(sequence\_path, sequence\_name + '/v/1.jpg') )

video\_writter\_v = cv2.VideoWriter(os.path.join(save\_sequence\_path, sequence\_name + '\_v.avi'), f, 25, (test\_frame\_v.shape[1], test\_frame\_v.shape[0]))

test\_frame\_i = cv2.imread(os.path.join(sequence\_path, sequence\_name + '/i/1.jpg') )

video\_writter\_i = cv2.VideoWriter(os.path.join(save\_sequence\_path, sequence\_name + '\_i.avi'), f, 25, (test\_frame\_i.shape[1], test\_frame\_i.shape[0]))

for channel in ['v', 'i']:

video\_path = os.path.join(sequence\_path, sequence\_name, channel)

label\_path = os.path.join(gt\_path, sequence\_name)

#for frame\_name in os.listdir(video\_path): # 这样读取是乱序，视频跳帧

# frame = cv2.imread(os.path.join(video\_path, frame\_name))

for index in range(len(os.listdir(video\_path))):

frame = cv2.imread(os.path.join( video\_path, str(index+1) + '.jpg') )

if channel == 'v':

now\_label = linecache.getline(os.path.join(label\_path, 'groundTruth\_v.txt'), index+1).strip() # 去掉换行符

else:

now\_label = linecache.getline(os.path.join(label\_path, 'groundTruth\_i.txt'), index+1).strip() # 去掉换行符

ori\_data\_seq\_vis\_path = os.path.join(label\_path, 'v')

ori\_data\_seq\_vis\_img\_name = os.listdir(ori\_data\_seq\_vis\_path)[0]

ori\_data\_seq\_vis\_img = cv2.imread(os.path.join(ori\_data\_seq\_vis\_path, ori\_data\_seq\_vis\_img\_name))

ratio\_w = test\_frame\_v.shape[1] / ori\_data\_seq\_vis\_img.shape[1]

ratio\_h = test\_frame\_v.shape[0] / ori\_data\_seq\_vis\_img.shape[0]

#print(now\_label)

[x, y, x1, y1] = now\_label.split(' ')

[x, y, x1, y1] = [int(x), int(y), int(x1), int(y1)]

cv2.rectangle(frame, ( int(x \* ratio\_w), int(y \* ratio\_h) ), ( int(x1 \* ratio\_w), int(y1 \* ratio\_h)), (255,255,255), 2)

if DEBUG:

cv2.imshow(sequence\_name + '\_' + channel, frame)

cv2.waitKey(20)

if channel == 'v':

video\_writter\_v.write(frame)

else:

video\_writter\_i.write(frame)

cv2.destroyAllWindows()

if \_\_name\_\_ == '\_\_main\_\_':

sequence\_path = './trackingResultsDisplay'

res\_video\_path = './my\_video'

gt\_path = 'xxx/dataset/GTOT'

gen\_video(sequence\_path, res\_video\_path, gt\_path)

* 脚本中注意的是上一步保存的图片大小与原图大小不一致，所以脚本中有缩放操作，运行gen\_video.py脚本之后，将在my\_video下得到对应序列名称的所有视频，其中白色框为gt框，红色框为APFNet算法的跟踪框。



附录：

1 gen\_video函数解析

for channel in ['v', 'i']:

……  
 cv2.destroyAllWindows()

这一段，出现了报错，下面对其进行详细分解：

这段代码的主要功能是将帧图像读取并加上矩形框，分别将视频的可见光通道（v）和红外通道（i）的帧写入视频文件。它从序列路径中加载图像，并从标签文件中获取标注信息，然后在帧图像中绘制对应的矩形框，最后生成带有标注的视频文件。

for channel in ['v', 'i']:

**遍历通道：**通道有两个，分别是 'v'（可见光）和 'i'（红外光），代码将遍历这两个通道，分别处理每个通道的图像。

video\_path = os.path.join(sequence\_path, sequence\_name, channel)

label\_path = os.path.join(gt\_path, sequence\_name)

**构造图像和标签路径：**

* video\_path：构造视频帧图像的路径，路径结构为 sequence\_path/sequence\_name/channel。
* label\_path：构造该序列的标签文件路径，用于从标签文件中读取每一帧的标注数据。

for index in range(len(os.listdir(video\_path))):

**循环遍历图像帧：**这段代码通过 os.listdir(video\_path) 获取视频帧目录下的所有文件，并使用 index 遍历这些文件。假设帧文件名是数字顺序的（如 1.jpg, 2.jpg），通过 index 递增逐帧处理。

frame = cv2.imread(os.path.join(video\_path, str(index + 1) + '.jpg'))

**读取当前帧图像：**使用 OpenCV 的 cv2.imread() 函数读取当前帧图像。文件名是通过 index + 1 动态生成的（因为帧文件名从 1.jpg 开始）。

if channel == 'v':

now\_label = linecache.getline(os.path.join(label\_path, 'groundTruth\_v.txt'), index + 1).strip()

else:

now\_label = linecache.getline(os.path.join(label\_path, 'groundTruth\_i.txt'), index + 1).strip()

**读取当前帧的标注信息：**

* 使用 linecache.getline() 读取标签文件中的第 index + 1 行，该行包含当前帧的标注数据（矩形框的坐标）。
* 根据通道不同，读取的文件分别是 groundTruth\_v.txt（可见光通道标签）或 groundTruth\_i.txt（红外通道标签）。

ori\_data\_seq\_vis\_path = os.path.join(label\_path, 'v')

ori\_data\_seq\_vis\_img\_name = os.listdir(ori\_data\_seq\_vis\_path)[0]

ori\_data\_seq\_vis\_img = cv2.imread(os.path.join(ori\_data\_seq\_vis\_path, ori\_data\_seq\_vis\_img\_name))

**读取可视化参考图像：**

* 这里读取可见光通道的某张参考图像，用于计算宽高缩放比例。它通过 os.listdir() 获取目录下的第一张图像，并使用 cv2.imread() 读取这张图像。

ratio\_w = test\_frame\_v.shape[1] / ori\_data\_seq\_vis\_img.shape[1]

ratio\_h = test\_frame\_v.shape[0] / ori\_data\_seq\_vis\_img.shape[0]

**计算缩放比例：**

* 通过参考图像和当前视频帧图像的宽高，计算横向（ratio\_w）和纵向（ratio\_h）的缩放比例。这是因为原始标注数据可能是基于参考图像的大小，而我们需要将这些标注缩放到当前帧图像的尺寸上。

[x, y, x1, y1] = now\_label.split(' ')

[x, y, x1, y1] = [int(x), int(y), int(x1), int(y1)]

cv2.rectangle(frame, (int(x \* ratio\_w), int(y \* ratio\_h)), (int(x1 \* ratio\_w), int(y1 \* ratio\_h)), (255, 255, 255), 2)

**解析标签并绘制矩形框：**

* now\_label 是一个字符串，包含矩形框的四个坐标点 (x, y, x1, y1)，通过 split() 将其分割成四个值。
* 这些值通过缩放比例进行放大/缩小，以适应当前帧的大小。
* 使用 cv2.rectangle() 在帧图像上绘制白色的矩形框，框的坐标为 (x, y) 和 (x1, y1)。

if DEBUG:

cv2.imshow(sequence\_name + '\_' + channel, frame)

cv2.waitKey(20)

**显示处理后的帧（调试模式）：**

如果 DEBUG 模式开启（DEBUG=1），则使用 cv2.imshow() 显示当前处理后的帧，并等待 20 毫秒。该代码用于实时查看处理效果。

if channel == 'v':

video\_writter\_v.write(frame)

else:

video\_writter\_i.write(frame)

**将帧写入视频文件：**

* 将处理后的帧图像写入对应的 VideoWriter 对象中，分别是可见光视频（video\_writter\_v）或红外光视频（video\_writter\_i）。

cv2.destroyAllWindows()

**销毁所有窗口**

**总结：**

* **输入**：每个序列的可见光通道和红外通道的图像帧，以及对应的标注数据。
* **输出**：为每个序列生成两个视频文件（一个为可见光通道，一个为红外通道），并在每帧图像上绘制矩形标注框。
* 通过对每帧进行标注和保存，最终生成标注视频。