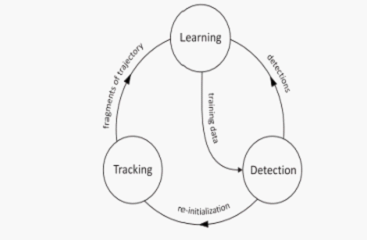
# [比微软kinect更强的视频跟踪算法--TLD跟踪算法介绍](http://blog.csdn.net/carson2005/article/details/7647500)

      TLD(Tracking-Learning-Detection)是英国萨里大学的一个捷克籍博士生在其攻读博士学位期间提出的一种新的单目标长时间（long term tracking）跟踪算法。该算法与传统跟踪算法的显著区别在于将传统的跟踪算法和传统的检测算法相结合来解决被跟踪目标在被跟踪过程中发生的形变、部分遮挡等问题。同时，通过一种改进的在线学习机制不断更新跟踪模块的“显著特征点”和检测模块的目标模型及相关参数，从而使得跟踪效果更加稳定、鲁棒、可靠。

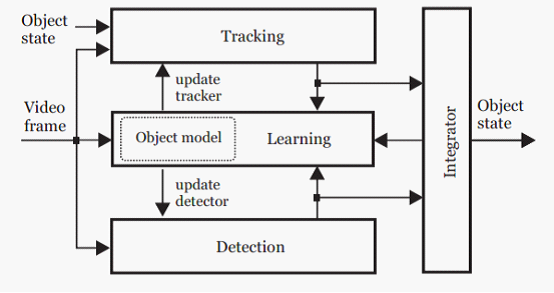
对于长时间跟踪而言，一个关键的问题是：当目标重新出现在相机视野中时，系统应该能重新检测到它，并开始重新跟踪。但是，长时间跟踪过程中，被跟踪目标将不可避免的发生形状变化、光照条件变化、尺度变化、遮挡等情况。传统的跟踪算法，前端需要跟检测模块相互配合，当检测到被跟踪目标之后，就开始进入跟踪模块，而此后，检测模块就不会介入到跟踪过程中。但这种方法有一个致命的缺陷：即，当被跟踪目标存在形状变化或遮挡时，跟踪就很容易失败；因此，对于长时间跟踪，或者被跟踪目标存在形状变化情况下的跟踪，很多人采用检测的方法来代替跟踪。该方法虽然在某些情况下可以改进跟踪效果，但它需要一个离线的学习过程。即：在检测之前，需要挑选大量的被跟踪目标的样本来进行学习和训练。这也就意味着，训练样本要涵盖被跟踪目标可能发生的各种形变和各种尺度、姿态变化和光照变化的情况。换言之，利用检测的方法来达到长时间跟踪的目的，对于训练样本的选择至关重要，否则，跟踪的鲁棒性就难以保证。

考虑到单纯的跟踪或者单纯的检测算法都无法在长时间跟踪过程中达到理想的效果，所以，TLD方法就考虑将两者予以结合，并加入一种改进的在线学习机制，从而使得整体的目标跟踪更加稳定、有效。

简单来说，TLD算法由三部分组成：跟踪模块、检测模块、学习模块；如下图所示

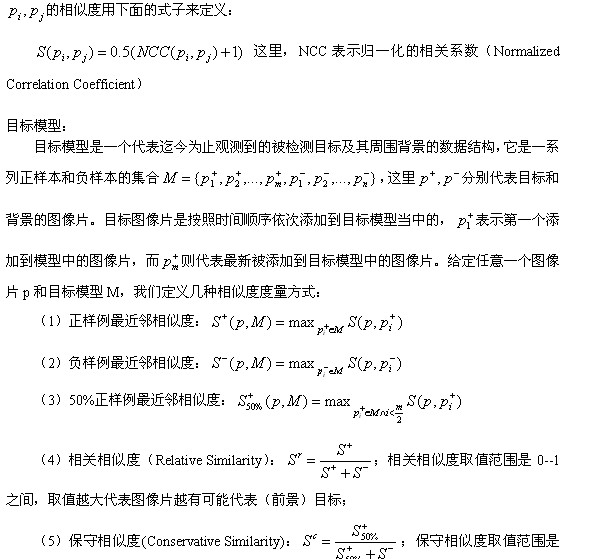


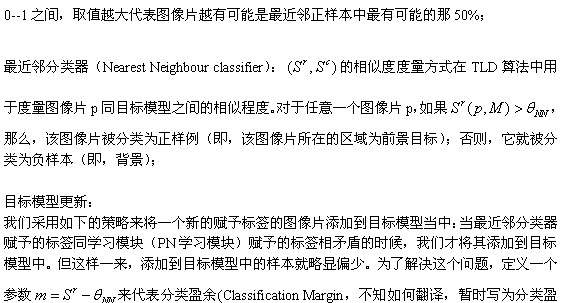
其运行机制为：检测模块和跟踪模块互不干涉的并行进行处理。首先，跟踪模块假设相邻视频帧之间物体的运动是有限的，且被跟踪目标是可见的，以此来估计目标的运动。如果目标在相机视野中消失，将造成跟踪失败。检测模块假设每一个视帧都是彼此独立的，并且根据以往检测和学习到的目标模型，对每一帧图片进行全图搜索以定位目标可能出现的区域。同其它目标检测方法一样，TLD中的检测模块也有可能出现错误，且错误无非是错误的负样例和错误的正样例这两种情况。而学习模块则根据跟踪模块的结果对检测模块的这两种错误进行评估，并根据评估结果生成训练样本对检测模块的目标模型进行更新，同时对跟踪模块的“关键特征点”进行更新，以此来避免以后出现类似的错误。TLD模块的详细；流程框图如下所示：

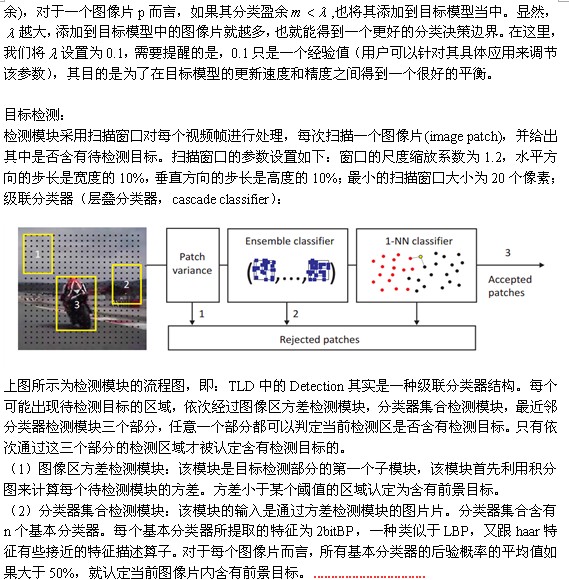


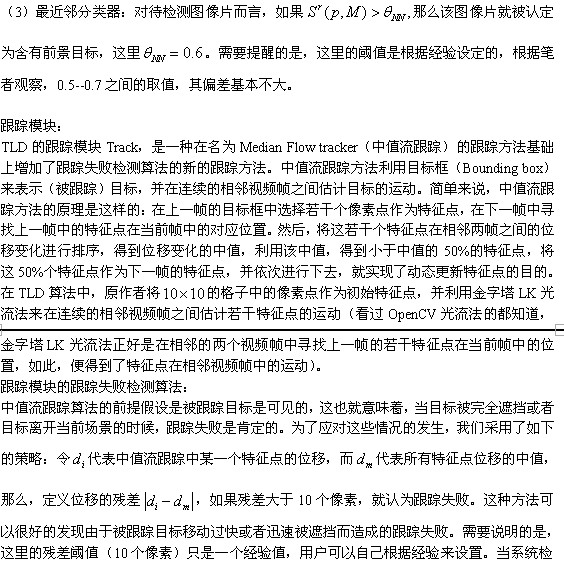
在详细介绍TLD的流程之前，有一些基本知识和基本概念需要予以澄清：

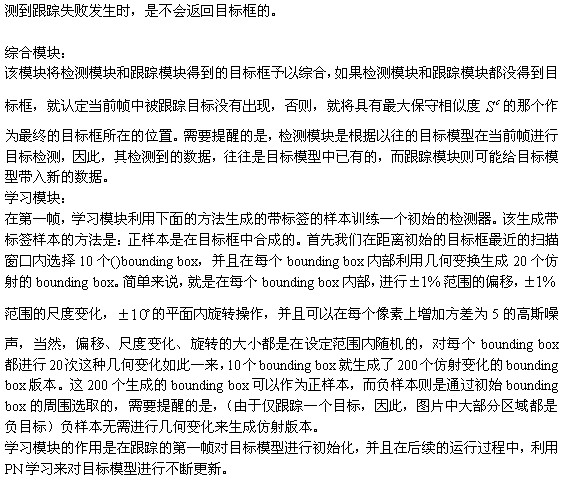
基本知识：

在任意时刻，被跟踪目标都可以用其状态属性来表示。该状态属性可以是一个表示目标所在位置、尺度大小的跟踪框，也可以是一个标识被跟踪目标是否可见的标记。两个跟踪框的空间域相似度是用重叠度（overlap）来度量，其计算方法是两个跟踪框的交集与两者并集的商。目标的形状采用图像片（image patch，个人认为，可以理解为滑动窗口）p来表示，每一个图像片都是从跟踪框内部采样得到的，并被归一化到15\*15的大小。两个图相片









# [OpenTLD 未完成 - 虎头](http://blog.csdn.net/windtalkersm/article/details/8018980)

TLD是一种算法的简称，原作者把它叫做Tracking-Learning-Detection。搞视觉的人看到这个名字都会吓一跳，很ambitious的计划。是09年的工作，不算太久，不过也不太新。网上关于这个的资源其实很多，很大程度和作者开放源代码有关。

学习过程中碰到的第一个问题就是资源太多---当然是相对这个领域而言，一般能找到一个忠实再现算法的源码就已经很好了。所以把找到的list一下，虽然有点浪费时间，希望可以对其他人有所帮助。具体的细节就不多说了，有很多很棒的分析也列在下面，比如[zouxy09](mailto:zouxy09@qq.com)写的源码注释，实在不能再详细了。如果硬要找茬，那就是大段的文字让人头晕，也没怎么排版。我倒想画几个简单的图补充一下，不知有什么好点的画图程序推荐(latex， or GNUPlot?没用过)

源代码资源：

1. 原作者 Zdenek Kalal

作者主页： http://info.ee.surrey.ac.uk/Personal/Z.Kalal/

源代码页: https://github.com/zk00006/OpenTLD

编程语言：Matlab + C

2. Alan Torres版

源代码页：https://github.com/alantrrs/OpenTLD

实现语言：C++

3. arthurv版

源代码页：https://github.com/arthurv/OpenTLD

实现语言：C++

注：和上面的没有发现任何区别

4. jmfs版

源代码页：https://github.com/jmfs/OpenTLD

实现语言：C++

注：和上面两个没有区别，只不过加入了VS2010工程文件，理论上可以直接在Windows下编译通过。不过opencv检测不到作者的webcam（！！！），所以他用了另一个VideoInput类来handle摄像头输入。

This is an adaptation of arthurv's fork of OpenTLD (<https://github.com/arthurv/OpenTLD>)

to be immeadiately runnable in Visual Studio 2010.

5. Georg Nebehay版 （终于有个不一样的了。。。。）

源代码页：http://gnebehay.github.com/OpenTLD/

注1：这个的好处是提供可执行文件下载（Ubuntu 10.04和Windows）。BUT, as you would expect，基本上到了你的机器上都跑不了。还是自己老老实实build吧。

注2：这个版本需要安装Qt。不过好像作者关掉了Qt的选项（相关代码还在），所以可以编译，但无法显示结果

注3：CSDN下载上有个“openTLD Qt 版“，就是这个版本。不过加了VS的工程文件---在我的机器上还是不能PnP， don't bother

http://download.csdn.net/download/muzi198783/4111915

6. Paul Nader版（又一个Qt 版！）

QOpenTLD: http://qopentld.sourceforge.net/

源代码页： http://sourceforge.net/projects/qopentld/

注1：需要OpenCV和Qt。 原系统要求Qt 4.3.7OpenCV 2.2。

注2：Windows和Linux下都提供了编译工程或makefile。估计也是唯一一个移植到Android平台下的TLD!

7. Ben Pryke版（又一个student project！）

源代码页：https://github.com/Ninjakannon/BPTLD

注：依然是Matlab+C/C++的混合实现。亮点是有很详细的Documentation（8页），介绍了算法的理解和实现细节。可以帮助理解原算法

博客资源（中文）：

1.  庖丁解牛TLD （yang\_xian521）

http://blog.csdn.net/yang\_xian521/article/details/6952870

注1:从文章看作者是基于原作者的matlab版分析的。从函数名看上面的2/3/4应该是matlab--->C++的"直译"，函数名都没变。这样最好，可以和下面的对照着看，同时学matlab和C++

2.  TLD（Tracking-Learning-Detection）学习与源码理解 (zouxy09)

http://blog.csdn.net/zouxy09/article/details/7893011

注1： 用的是<<arthurv版>>，前面说过，不能再详细了！

注2： 下面三个是从这篇copy的

3. 《再谈PN学习》：  
http://blog.csdn.net/carson2005/article/details/7647519  
4. 《比微软kinect更强的视频跟踪算法--TLD跟踪算法介绍》  
http://blog.csdn.net/carson2005/article/details/7647500  
5. 《TLD视觉跟踪技术解析》  
http://www.asmag.com.cn/number/n-50168.shtml

想说的话：

1. 分享：前段时间把 TLD::init(...)看完了，本想一鼓作气，其他的事太多只好放下。不过我对detection和tracking比较熟，init中已经把learning作了一遍，看懂了剩下的就容易了。现在重新捡起，无意中发现了zouxy09的注释，省了太多力气，半天就看完了。很多细节不用自己去抠--- 我们常抱怨这资源那文档太少，羡慕老外能力强，动作快，和他们愿意分享关系太大。经常看到一些不错的文章收藏起来，过几天去看居然删了！

可以理解可能是开公司要保密，但如果害怕别人知道了自己的思路就做不下去，那还是不要在这个领域做了。算法只是思想，谁也垄断不了。算法也一定是不断更新的， 抱着一个算法不放也存活不了几年。原作者也基于这个技术开公司了，也没见他们基于这个限制别人使用。SIFT，SURF都patent了也没听说赚了大钱，kinect告诉你算法也实现不了。要保密的是实现细节

2. 比较： 终于看完了实现，总的感觉这个算法还是更像工程应用（engineering）而不是理论突破（也不能要求太多了是不是）。感觉这么结合后并不一定会比单个的跟踪（tracking）或检测（detection-by-classification）模块作的更好，毕竟还是没有解决外观(appearence)和尺度(scale)变化这两个根本难题。 不过这种框架反而应该在实际中非常实用，因为----------可调的参数太多了!

TLD相信很多人都试过了，实时性很多人都在抱怨，而且拿到自己的视频上总要调些参数效果才好。

比较起来更喜欢今年ECCV上Kaihua Zhang的Compressive Tracking：理论高深的吓人（开玩笑），源码简单的吓人。而且是目前为止我试过的off-the-shelf的tracker中跟踪效果最好的，不用调任何参数，绝对实时----代码那么少，想不实时都难吧（顺便说一句作者的blog就在上面提到过）。这才是做研究的方法，有个强大的理论做支撑，实现可以很简单却不会影响效果。所以如果搞数学的人愿意做应用，很多人都会下岗

http://www4.comp.polyu.edu.hk/~cslzhang/CT/CT.htm

另一个PWP（Pixel-Wise Posteriors），publish时间和TLD差不多，性能看上去也很美，不过作者说要开源，一直没有兑现。是个遗憾。个人觉得level set对部分遮挡效果应该很好，做到实时也不是难事

http://www.robots.ox.ac.uk/~cbibby/research\_pwp.shtml

3. 总结：TLD其实是一个非常合适的入门和进阶算法：

a. 有理论，有高质量的paper（BMVC, CVPR, ICPR, 最后PAMI）

b. 有源代码！Matlab， C++， Windows, Linux, .....你还想要啥？

c. 有不同大牛小牛分享的详细的介绍和详细的代码注释（几乎每一行都解释到了）！

4. 牵涉面广，涉及到detection, tracking, classifcation，传统的视觉技术就是这么硬梆梆的划分的三大类。研究完了对每一部分多少能有点心得。

The End

TLD跟踪算法优化（一）并行化

才学疏浅，只言片语，只求志同道的朋友一起交流研究。

并行化不算是算法的改进，只是追求运行的实时性。

简要列举一个例子：

TLD算法的C++版本源码里：

LKTracker::trackf2f(const Mat& img1, const Mat& img2,vector<Point2f> &points1, vector<cv::Point2f> &points2){

[cpp] view plaincopy

1. bool LKTracker::trackf2f(const Mat& img1, const Mat& img2,vector<Point2f> &points1, vector<cv::Point2f> &points2){

2. //TODO!:implement c function cvCalcOpticalFlowPyrLK() or Faster tracking function

3. //Forward-Backward tracking

4. #pragma omp parallel sections //声明该并行区域分为若干个section,section之间的运行顺序为并行的关系

5. {

6. #pragma omp section //第一个section,由某个线程单独完成

7. //前向轨迹跟踪

8. calcOpticalFlowPyrLK( img1,img2, points1, points2, status,similarity, window\_size, level, term\_criteria, lambda, 0);

9.

10.

11. #pragma omp section //第二个section,由某个线程单独完成

12. //后向轨迹跟踪

13. calcOpticalFlowPyrLK( img2,img1, points2, pointsFB, FB\_status,FB\_error, window\_size, level, term\_criteria, lambda, 0);

14. }

15. //前向轨迹跟踪

16. // calcOpticalFlowPyrLK( img1,img2, points1, points2, status,similarity, window\_size, level, term\_criteria, lambda, 0);

17. //后向轨迹跟踪

18. //calcOpticalFlowPyrLK( img2,img1, points2, pointsFB, FB\_status,FB\_error, window\_size, level, term\_criteria, lambda, 0);

19.

20. //Compute the real FB-error

21. /\*

22. 原理很简单：从t时刻的图像的A点，跟踪到t+1时刻的图像B点；然后倒回来

23. 从t+1时刻的图像的B点往回跟踪，假如跟踪到t时刻的图像的C点，这样就产

24. 生了前向和后向两个轨迹，比较t时刻中A点和C点的距离，如果距离小于某个

25. 阈值，那么就认为前向跟踪是正确的；这个距离就是FB\_error

26. \*/

27. //计算前向与后向轨迹的误差。

28. #pragma omp parallel for

29. for( int i= 0; i<points1.size(); ++i ){

30. FB\_error[i] = norm(pointsFB[i]-points1[i]); //norm求矩阵或向量的

31. //范数,或绝对值

32. }

33. //Filter out points with FB\_error[i] > median(FB\_error) && points with sim\_error[i] > median(sim\_error)

34. normCrossCorrelation(img1,img2,points1,points2);

35. return filterPts(points1,points2);

36. }

修改后代码运行速度提高了不少。

不过并行化处理，必须考虑到一些问题

1.数据的互斥问题

2.线程的分配问题

3.Release版本应用程序对于for循环可以自动优化，不用对for做多线程设定，主要还是放在模块化的数据处理并行化上。

# [openTLD 源码解读](http://blog.csdn.net/muzi198783/article/details/7392255)

首先是run\_TLD 在其次就是tldExample 最后到了初始化函数tldInit

第一个比较关键的函数   bb\_scan

将图像网格化，将图片首先 SCALE = 1.2.^[-10:10];（21 个规格），在每个规格上打网格

这个函数有一个比较重要的方法

ntuples（就是重复） 因为网格上的点很多点有相同的X，或者Y 具体方法大家在matlab中调用一句ntuples([1 2],[1 2,3])看看答案就知道什么意思了。

这个函数返回一个6\*n的矩阵，这个矩阵每个列代表一个窗口。

1，代表窗口左上点X

2，代表窗口左上点y

3，代表窗口右下点X

4，代表窗口右下点Y

5，这种规格的编号

6，相同X的窗口数目

产生特征点函数tldGenerateFeatures

就是产生一系列的随机数 大概是（4\*特征个数）\*树的个数

初始化轨迹

记录连续的图片

训练探测器

规格化和目标的相似度bb\_overlap

将每个规格化窗口和目标窗口做运算得到相似度

公式：相交的面积/总面积(不相交则返回0.0)

注意矩阵运算返回一个向量，大小就是规格化的数目length（grid）

取出图片的函数，当参数为4个时会进行变换img\_patch

tldGeneratePositiveData产生正样本数据

主要利用的是相似度来判定正负样本既bb\_overlap>阈值，则认为正样本，反之亦然。

这里主要就是有个编码问题

随机取出点比较大小进行编码

for (int i=0; i<nFEAT; i++) {  
index<<=1;   
int fp0 = img[off[0]+bbox[0]];  
int fp1 = img[off[1]+bbox[0]];  
if (fp0>fp1) { index |= 1;}  
off += 2;  
}

(fern.cpp的measure\_tree\_offset)

这个函数返回一个pX代表编码后的矩阵，pEx代表最靠近目标窗口的规格化窗口（这个窗口进行了处理可以用于后面的训练），

最靠近目标窗口的规格化窗口

类似的负样本情况。

fern(2,......)

开始训练随机树

if (Y[I] == 1) {  
if (measure\_forest(x) <= thrP)//防止某个特征占的比重太大  
update(x,1,1);  
} else {  
if (measure\_forest(x) >= thrN)  
update(x,0,1);

void update(double \*x, int C, int N) {  
for (int i = 0; i < nTREES; i++) {  
int idx = (int) x[i];  
(C==1) ? nP[i][idx] += N : nN[i][idx] += N;  
if (nP[i][idx]==0) {  
WEIGHT[i][idx] = 0;  
} else {  
WEIGHT[i][idx] = ((double) (nP[i][idx])) / (nP[i][idx] + nN[i][idx]);//正样本增加相应位置的权重，集后面的置信度  
}  
}  
}

训练样本产生一个正确的特征集

 tldTrainNN(pEx,nEx,tld)

tld.pex = [tld.pex(:,1:isin(2)) x(:,i) tld.pex(:,isin(2)+1:end)]; % 这个代码就是通过随机树种产生比较高的规格化窗口在这些窗口放到正特征集里面

以后就可以拿这个和目标最比较来判断目标窗口的位置

[TLD（Tracking-Learning-Detection）学习与源码理解](http://blog.csdn.net/zouxy09/article/details/7893011)

## 引言

         TLD(Tracking-Learning-Detection)是英国萨里大学的一个捷克籍博士生Zdenek Kalal在其攻读博士学位期间提出的一种新的单目标长时间（long term tracking）跟踪算法。该算法与传统跟踪算法的显著区别在于将传统的跟踪算法和传统的检测算法相结合来解决被跟踪目标在被跟踪过程中发生的形变、部分遮挡等问题。同时，通过一种改进的在线学习机制不断更新跟踪模块的“显著特征点”和检测模块的目标模型及相关参数，从而使得跟踪效果更加稳定、鲁棒、可靠。

       作者网站的链接<http://info.ee.surrey.ac.uk/Personal/Z.Kalal/>

       其开放源代码，在网站上可以下载到源代码已经其demo程序，但是源代码是由Matlab和C写的，对于我这种不懂Matlab的菜鸟来说，看代码就像天书；但很庆幸，有一个大牛已经用c++把TLD重新写好了，而且代码很规范。并且提供源码下载：

[https://github.com/arthurv/OpenTLD](http://www.google.com/url?sa=D&q=https://github.com/arthurv/OpenTLD&usg=AFQjCNGDXLUFLgmCuf4Np-Tr6241h_G6fA)

      源码为Linux版本，基于Opencv2.3在源码/doc文件夹下有其程序设计接口，很清晰。

ZK关于这个TLD框架发表了很多论文，感觉对理解代码非常有用的论文有下面三个：

（1）Tracking-Learning-Detection

（2）Forward-Backward Error Automatic Detection of Tracking Failures

（3）Online learning of robust object detectors during unstable tracking

在作者的网站上好像也提供下载了。

       另外自己学习的过程中，也搜到了不少大牛对TLD的分析，得到了很多帮助，具体有：

（1）《庖丁解牛TLD》系列：

<http://blog.csdn.net/yang_xian521/article/details/7091587>

（2）《再谈PN学习》：

<http://blog.csdn.net/carson2005/article/details/7647519>

（3）《比微软kinect更强的视频跟踪算法--TLD跟踪算法介绍》

<http://blog.csdn.net/carson2005/article/details/7647500>

（4）《TLD视觉跟踪技术解析》

<http://www.asmag.com.cn/number/n-50168.shtml>

       自己在看论文和这些大牛的分析过程中，对代码进行了一些理解，但是由于自己接触图像处理和机器视觉没多久，另外由于自己编程能力比较弱，所以分析过程可能会有不少的错误，希望各位不吝指正。具体代码分析见博客的更新。

## OpenTLD下载与编译：

（1）<https://github.com/arthurv/OpenTLD>  
下载得到：arthurv-OpenTLD-1e3cd0b.zip

或者在Linux下直接通过git工具进行克隆：

#git clone git@github.com:alantrrs/OpenTLD.git

（2）我的编译环境是Ubuntu 12.04 + Opencv2.3

安装opencv 2.3：

#apt-get install libcv-dev libcv2.3 libcvaux-dev libcvaux2.3 libhighgui-dev libhighgui2.3

安装cmake：

#sudo apt-get install cmake

解压然后按照源码目录下README文件进行编译：

#cd OpenTLD

#mkdir build

#cd build

#cmake ../src/

#make

#cd ../bin/

（3）运行：

%To run from camera

./run\_tld -p ../parameters.yml

%To run from file

./run\_tld -p ../parameters.yml -s ../datasets/06\_car/car.mpg

%To init bounding box from file

./run\_tld -p ../parameters.yml -s ../datasets/06\_car/car.mpg -b ../datasets/06\_car/init.txt

%To train only in the firs frame (no tracking, no learning)

./run\_tld -p ../parameters.yml -s ../datasets/06\_car/car.mpg -b ../datasets/06\_car/init.txt -no\_tl

%To test the final detector (Repeat the video, first time learns, second time detects)

./run\_tld -p ../parameters.yml -s ../datasets/06\_car/car.mpg -b ../datasets/06\_car/init.txt –r

## Main函数

       下面是自己在看论文和这些大牛的分析过程中，对代码进行了一些理解，但是由于自己接触图像处理和机器视觉没多久，另外由于自己编程能力比较弱，所以分析过程可能会有不少的错误，希望各位不吝指正。而且，因为编程很多地方不懂，所以注释得非常乱，还海涵。

从main()函数切入，分析整个TLD运行过程如下：

（这里只是分析工作过程，全部注释的代码见博客的更新）

1、分析程序运行的命令行参数；

./run\_tld -p ../parameters.yml -s ../datasets/06\_car/car.mpg -b ../datasets/06\_car/init.txt –r

2、读入初始化参数（程序中变量）的文件parameters.yml；

3、通过文件或者用户鼠标框选的方式指定要跟踪的目标的Bounding Box；

4、用上面得到的包含要跟踪目标的Bounding  Box和第一帧图像去初始化TLD系统，

   tld.init(last\_gray, box, bb\_file); 初始化包含的工作如下：

4.1、buildGrid(frame1, box);

检测器采用扫描窗口的策略：扫描窗口步长为宽高的 10%，尺度缩放系数为1.2；此函数构建全部的扫描窗口grid，并计算每一个扫描窗口与输入的目标box的重叠度；重叠度定义为两个box的交集与它们的并集的比；

4.2、为各种变量或者容器分配内存空间；

4.3、getOverlappingBoxes(box, num\_closest\_init);

此函数根据传入的box（目标边界框），在整帧图像中的全部扫描窗口中（由上面4.1得到）寻找与该box距离最小（即最相似，重叠度最大）的num\_closest\_init（10）个窗口，然后把这些窗口归入good\_boxes容器。同时，把重叠度小于0.2的，归入bad\_boxes容器；相当于对全部的扫描窗口进行筛选。并通过BBhull函数得到这些扫描窗口的最大边界。

4.5、classifier.prepare(scales);

准备分类器，scales容器里是所有扫描窗口的尺度，由上面的buildGrid()函数初始化；

TLD的分类器有三部分：方差分类器模块、集合分类器模块和最近邻分类器模块；这三个分类器是级联的，每一个扫描窗口依次全部通过上面三个分类器，才被认为含有前景目标。这里prepare这个函数主要是初始化集合分类器模块；

集合分类器（随机森林）基于n个基本分类器（共10棵树），每个分类器（树）都是基于一个pixel comparisons（共13个像素比较集）的，也就是说每棵树有13个判断节点（组成一个pixel comparisons），输入的图像片与每一个判断节点（相应像素点）进行比较，产生0或者1，然后将这13个0或者1连成一个13位的二进制码x（有2^13种可能），每一个x对应一个后验概率P(y|x)= #p/(#p+#n) （也有2^13种可能），#p和#n分别是正和负图像片的数目。那么整一个集合分类器（共10个基本分类器）就有10个后验概率了，将10个后验概率进行平均，如果大于阈值（一开始设经验值0.65，后面再训练优化）的话，就认为该图像片含有前景目标；

后验概率P(y|x)= #p/(#p+#n)的产生方法：初始化时，每个后验概率都得初始化为0；运行时候以下面方式更新：将已知类别标签的样本（训练样本）通过n个分类器进行分类，如果分类结果错误，那么相应的#p和#n就会更新，这样P(y|x)也相应更新了。

pixel comparisons的产生方法：先用一个归一化的patch去离散化像素空间，产生所有可能的垂直和水平的pixel comparisons，然后我们把这些pixel comparisons随机分配给n个分类器，每个分类器得到完全不同的pixel comparisons（特征集合），这样，所有分类器的特征组统一起来就可以覆盖整个patch了。

特征是相对于一种尺度的矩形框而言的，TLD中第s种尺度的第i个特征features[s][i] = Feature(x1, y1, x2, y2);是两个随机分配的像素点坐标（就是由这两个像素点比较得到0或者1的）。每一种尺度的扫描窗口都含有totalFeatures = nstructs \* structSize个特征；nstructs为树木（由一个特征组构建，每组特征代表图像块的不同视图表示）的个数；structSize为每棵树的特征个数，也即每棵树的判断节点个数；树上每一个特征都作为一个决策节点；

prepare函数的工作就是先给每一个扫描窗口初始化了对应的pixel comparisons（两个随机分配的像素点坐标）；然后初始化后验概率为0；

4.6、generatePositiveData(frame1, num\_warps\_init);

此函数通过对第一帧图像的目标框box（用户指定的要跟踪的目标）进行仿射变换来合成训练初始分类器的正样本集。具体方法如下：先在距离初始的目标框最近的扫描窗口内选择10个bounding box（已经由上面的getOverlappingBoxes函数得到，存于good\_boxes里面了，还记得不？），然后在每个bounding box的内部，进行±1%范围的偏移，±1%范围的尺度变化，±10%范围的平面内旋转，并且在每个像素上增加方差为5的高斯噪声（确切的大小是在指定的范围内随机选择的），那么每个box都进行20次这种几何变换，那么10个box将产生200个仿射变换的bounding box，作为正样本。具体实现如下：

getPattern(frame(best\_box), pEx, mean, stdev);此函数将frame图像best\_box区域的图像片归一化为均值为0的15\*15大小的patch，存于pEx（用于最近邻分类器的正样本）正样本中（最近邻的box的Pattern），该正样本只有一个。

generator(frame, pt, warped, bbhull.size(), rng);此函数属于PatchGenerator类的构造函数，用来对图像区域进行仿射变换，先RNG一个随机因子，再调用（）运算符产生一个变换后的正样本。

classifier.getFeatures(patch, grid[idx].sidx, fern);函数得到输入的patch的特征fern（13位的二进制代码）；

pX.push\_back(make\_pair(fern, 1));   //positive ferns <features, labels=1>然后标记为正样本，存入pX（用于集合分类器的正样本）正样本库；

以上的操作会循环 num\_warps \* good\_boxes.size()即20 \* 10 次，这样，pEx就有了一个正样本，而pX有了200个正样本了；

4.7、meanStdDev(frame1(best\_box), mean, stdev);

统计best\_box的均值和标准差，var = pow(stdev.val[0],2) \* 0.5;作为方差分类器的阈值。

4.8、generateNegativeData(frame1);

     由于TLD仅跟踪一个目标，所以我们确定了目标框了，故除目标框外的其他图像都是负样本，无需仿射变换；具体实现如下：

     由于之前重叠度小于0.2的，都归入 bad\_boxes了，所以数量挺多，把方差大于var\*0.5f的bad\_boxes都加入负样本，同上面一样，需要classifier.getFeatures(patch, grid[idx].sidx, fern);和nX.push\_back(make\_pair(fern, 0));得到对应的fern特征和标签的nX负样本（用于集合分类器的负样本）；

    然后随机在上面的bad\_boxes中取bad\_patches（100个）个box，然后用 getPattern函数将frame图像bad\_box区域的图像片归一化到15\*15大小的patch，存在nEx（用于最近邻分类器的负样本）负样本中。

这样nEx和nX都有负样本了；（box的方差通过积分图像计算）

4.9、然后将nEx的一半作为训练集nEx，另一半作为测试集nExT；同样，nX也拆分为训练集nX和测试集nXT；

4.10、将负样本nX和正样本pX合并到ferns\_data[]中，用于集合分类器的训练；

4.11、将上面得到的一个正样本pEx和nEx合并到nn\_data[]中，用于最近邻分类器的训练；

4.12、用上面的样本训练集训练 集合分类器（森林） 和 最近邻分类器：

  classifier.trainF(ferns\_data, 2); //bootstrap = 2

对每一个样本ferns\_data[i] ，如果样本是正样本标签，先用measure\_forest函数返回该样本所有树的所有特征值对应的后验概率累加值，该累加值如果小于正样本阈值（0.6\* nstructs，这就表示平均值需要大于0.6（0.6\* nstructs / nstructs）,0.6是程序初始化时定的集合分类器的阈值，为经验值，后面会用测试集来评估修改，找到最优），也就是输入的是正样本，却被分类成负样本了，出现了分类错误，所以就把该样本添加到正样本库，同时用update函数更新后验概率。对于负样本，同样，如果出现负样本分类错误，就添加到负样本库。

  classifier.trainNN(nn\_data);

     对每一个样本nn\_data，如果标签是正样本，通过NNConf(nn\_examples[i], isin, conf, dummy);计算输入图像片与在线模型之间的相关相似度conf，如果相关相似度小于0.65 ，则认为其不含有前景目标，也就是分类错误了；这时候就把它加到正样本库。然后就通过pEx.push\_back(nn\_examples[i]);将该样本添加到pEx正样本库中；同样，如果出现负样本分类错误，就添加到负样本库。

4.13、用测试集在上面得到的 集合分类器（森林） 和 最近邻分类器中分类，评价并修改得到最好的分类器阈值。

  classifier.evaluateTh(nXT, nExT);

   对集合分类器，对每一个测试集nXT，所有基本分类器的后验概率的平均值如果大于thr\_fern（0.6），则认为含有前景目标，然后取最大的平均值（大于thr\_fern）作为该集合分类器的新的阈值。

   对最近邻分类器，对每一个测试集nExT，最大相关相似度如果大于nn\_fern（0.65），则认为含有前景目标，然后取最大的最大相关相似度（大于nn\_fern）作为该最近邻分类器的新的阈值。

5、进入一个循环：读入新的一帧，然后转换为灰度图像，然后再处理每一帧processFrame；

6、processFrame(last\_gray, current\_gray, pts1, pts2, pbox, status, tl, bb\_file);逐帧读入图片序列，进行算法处理。processFrame共包含四个模块（依次处理）：跟踪模块、检测模块、综合模块和学习模块；

6.1、跟踪模块：track(img1, img2, points1, points2);

track函数完成前一帧img1的特征点points1到当前帧img2的特征点points2的跟踪预测；

6.1.1、具体实现过程如下：

（1）先在lastbox中均匀采样10\*10=100个特征点（网格均匀撒点），存于points1：

bbPoints(points1, lastbox);

（2）利用金字塔LK光流法跟踪这些特征点，并预测当前帧的特征点（见下面的解释）、计算FB error和匹配相似度sim，然后筛选出 FB\_error[i] <= median(FB\_error) 和 sim\_error[i] > median(sim\_error) 的特征点（舍弃跟踪结果不好的特征点），剩下的是不到50%的特征点：

tracker.trackf2f(img1, img2, points, points2);

（3）利用剩下的这不到一半的跟踪点输入来预测bounding box在当前帧的位置和大小 tbb：

bbPredict(points, points2, lastbox, tbb);

（4）跟踪失败检测：如果FB error的中值大于10个像素（经验值），或者预测到的当前box的位置移出图像，则认为跟踪错误，此时不返回bounding box：

if (tracker.getFB()>10 || tbb.x>img2.cols ||  tbb.y>img2.rows || tbb.br().x < 1 || tbb.br().y <1)

（5）归一化img2(bb)对应的patch的size（放缩至patch\_size = 15\*15），存入pattern：

getPattern(img2(bb),pattern,mean,stdev);

（6）计算图像片pattern到在线模型M的保守相似度：

classifier.NNConf(pattern,isin,dummy,tconf);

（7）如果保守相似度大于阈值，则评估本次跟踪有效，否则跟踪无效：

if (tconf>classifier.thr\_nn\_valid) tvalid =true;

6.1.2、TLD跟踪模块的实现原理和trackf2f函数的实现：

   TLD跟踪模块的实现是利用了Media Flow 中值光流跟踪和跟踪错误检测算法的结合。中值流跟踪方法是基于Forward-Backward Error和NNC的。原理很简单：从t时刻的图像的A点，跟踪到t+1时刻的图像B点；然后倒回来，从t+1时刻的图像的B点往回跟踪，假如跟踪到t时刻的图像的C点，这样就产生了前向和后向两个轨迹，比较t时刻中 A点和C点的距离，如果距离小于一个阈值，那么就认为前向跟踪是正确的；这个距离就是FB\_error；

bool LKTracker::trackf2f(const Mat& img1, const Mat& img2, vector<Point2f> &points1, vector<cv::Point2f> &points2)

函数实现过程如下：

（1）先利用金字塔LK光流法跟踪预测前向轨迹：

  calcOpticalFlowPyrLK( img1,img2, points1, points2, status, similarity, window\_size, level, term\_criteria, lambda, 0);

（2）再往回跟踪，产生后向轨迹：

  calcOpticalFlowPyrLK( img2,img1, points2, pointsFB, FB\_status,FB\_error, window\_size, level, term\_criteria, lambda, 0);

（3）然后计算 FB-error：前向与 后向 轨迹的误差：

  for( int i= 0; i<points1.size(); ++i )

        FB\_error[i] = norm(pointsFB[i]-points1[i]);

（4）再从前一帧和当前帧图像中（以每个特征点为中心）使用亚象素精度提取10x10象素矩形（使用函数getRectSubPix得到），匹配前一帧和当前帧中提取的10x10象素矩形，得到匹配后的映射图像（调用matchTemplate），得到每一个点的NCC相关系数（也就是相似度大小）。

normCrossCorrelation(img1, img2, points1, points2);

（5）然后筛选出 FB\_error[i] <= median(FB\_error) 和 sim\_error[i] > median(sim\_error) 的特征点（舍弃跟踪结果不好的特征点），剩下的是不到50%的特征点；

filterPts(points1, points2);

6.2、检测模块：detect(img2);

TLD的检测分类器有三部分：方差分类器模块、集合分类器模块和最近邻分类器模块；这三个分类器是级联的。当前帧img2的每一个扫描窗口依次通过上面三个分类器，全部通过才被认为含有前景目标。具体实现过程如下：

先计算img2的积分图，为了更快的计算方差：

integral(frame,iisum,iisqsum);

然后用高斯模糊，去噪：

  GaussianBlur(frame,img,Size(9,9),1.5);

下一步就进入了方差检测模块：

6.2.1、方差分类器模块：getVar(grid[i],iisum,iisqsum) >= var

利用积分图计算每个待检测窗口的方差，方差大于var阈值（目标patch方差的50%）的，则认为其含有前景目标，通过该模块的进入集合分类器模块：

6.2.2、集合分类器模块：

集合分类器（随机森林）共有10颗树（基本分类器），每棵树13个判断节点，每个判断节点经比较得到一个二进制位0或者1，这样每棵树就对应得到一个13位的二进制码x（叶子），这个二进制码x对应于一个后验概率P(y|x)。那么整一个集合分类器（共10个基本分类器）就有10个后验概率了，将10个后验概率进行平均，如果大于阈值（一开始设经验值0.65，后面再训练优化）的话，就认为该图像片含有前景目标；具体过程如下：

（1）先得到该patch的特征值（13位的二进制代码）：

classifier.getFeatures(patch,grid[i].sidx,ferns);

（2）再计算该特征值对应的后验概率累加值：

conf = classifier.measure\_forest(ferns);

（3）若集合分类器的后验概率的平均值大于阈值fern\_th（由训练得到），就认为含有前景目标：

if (conf > numtrees \* fern\_th)  dt.bb.push\_back(i);

（4）将通过以上两个检测模块的扫描窗口记录在detect structure中；

（5）如果顺利通过以上两个检测模块的扫描窗口数大于100个，则只取后验概率大的前100个；

nth\_element(dt.bb.begin(), dt.bb.begin()+100, dt.bb.end(),

CComparator(tmp.conf));

进入最近邻分类器：

6.2.3、最近邻分类器模块

（1）先归一化patch的size（放缩至patch\_size = 15\*15），存入dt.patch[i];

getPattern(patch,dt.patch[i],mean,stdev);

（2）计算图像片pattern到在线模型M的相关相似度和保守相似度：

classifier.NNConf(dt.patch[i],dt.isin[i],dt.conf1[i],dt.conf2[i]);

（3）相关相似度大于阈值，则认为含有前景目标：

if (dt.conf1[i]>nn\_th)  dbb.push\_back(grid[idx]);

到目前为止，检测器检测完成，全部通过三个检测模块的扫描窗口存在dbb中；

6.3、综合模块：

TLD只跟踪单目标，所以综合模块综合跟踪器跟踪到的单个目标和检测器可能检测到的多个目标，然后只输出保守相似度最大的一个目标。具体实现过程如下：

（1）先通过 重叠度 对检测器检测到的目标bounding box进行聚类，每个类的重叠度小于0.5：

clusterConf(dbb, dconf, cbb, cconf);

（2）再找到与跟踪器跟踪到的box距离比较远的类（检测器检测到的box），而且它的相关相似度比跟踪器的要大：记录满足上述条件，也就是可信度比较高的目标box的个数：

if (bbOverlap(tbb, cbb[i])<0.5 && cconf[i]>tconf) confident\_detections++;

（3）判断如果只有一个满足上述条件的box，那么就用这个目标box来重新初始化跟踪器（也就是用检测器的结果去纠正跟踪器）：

if (confident\_detections==1)  bbnext=cbb[didx];

（4）如果满足上述条件的box不只一个，那么就找到检测器检测到的box与跟踪器预测到的box距离很近（重叠度大于0.7）的所以box，对其坐标和大小进行累加：

if(bbOverlap(tbb,dbb[i])>0.7)  cx += dbb[i].x;……

（5）对与跟踪器预测到的box距离很近的box 和 跟踪器本身预测到的box 进行坐标与大小的平均作为最终的目标bounding box，但是跟踪器的权值较大：

bbnext.x = cvRound((float)(10\*tbb.x+cx)/(float)(10+close\_detections));……

（6）另外，如果跟踪器没有跟踪到目标，但是检测器检测到了一些可能的目标box，那么同样对其进行聚类，但只是简单的将聚类的cbb[0]作为新的跟踪目标box（不比较相似度了？？还是里面已经排好序了？？），重新初始化跟踪器：

bbnext=cbb[0];

至此，综合模块结束。

6.4、学习模块：learn(img2);

    学习模块也分为如下四部分：

6.4.1、检查一致性：

（1）归一化img(bb)对应的patch的size（放缩至patch\_size = 15\*15），存入pattern：

  getPattern(img(bb), pattern, mean, stdev);

（2）计算输入图像片（跟踪器的目标box）与在线模型之间的相关相似度conf：

  classifier.NNConf(pattern,isin,conf,dummy);

（3）如果相似度太小了或者如果方差太小了或者如果被被识别为负样本，那么就不训练了；

if (conf<0.5)……或if (pow(stdev.val[0], 2)< var)……或if(isin[2]==1)……

6.4.2、生成样本：

先是集合分类器的样本：fern\_examples：

（1）先计算所有的扫描窗口与目前的目标box的重叠度：

grid[i].overlap = bbOverlap(lastbox, grid[i]);

（2）再根据传入的lastbox，在整帧图像中的全部窗口中寻找与该lastbox距离最小（即最相似，重叠度最大）的num\_closest\_update个窗口，然后把这些窗口归入good\_boxes容器（只是把网格数组的索引存入）同时，把重叠度小于0.2的，归入 bad\_boxes 容器：

  getOverlappingBoxes(lastbox, num\_closest\_update);

（3）然后用仿射模型产生正样本（类似于第一帧的方法，但只产生10\*10=100个）：

generatePositiveData(img, num\_warps\_update);

（4）加入负样本，相似度大于1？？相似度不是出于0和1之间吗？

idx=bad\_boxes[i];

if (tmp.conf[idx]>=1) fern\_examples.push\_back(make\_pair(tmp.patt[idx],0));

然后是最近邻分类器的样本：nn\_examples：

if (bbOverlap(lastbox,grid[idx]) < bad\_overlap)

        nn\_examples.push\_back(dt.patch[i]);

6.4.3、分类器训练：

classifier.trainF(fern\_examples,2);

classifier.trainNN(nn\_examples);

6.4.4、把正样本库（在线模型）包含的所有正样本显示在窗口上

classifier.show();

至此，tld.processFrame函数结束。

7、如果跟踪成功，则把相应的点和box画出来：

    if (status){

      drawPoints(frame,pts1);

      drawPoints(frame,pts2,Scalar(0,255,0));  //当前的特征点用蓝色点表示

      drawBox(frame,pbox);

      detections++;

}

8、然后显示窗口和交换图像帧，进入下一帧的处理：

    imshow("TLD", frame);

swap(last\_gray, current\_gray);

至此，main()函数结束（只分析了框架）。

## run\_tld

       下面是自己在看论文和这些大牛的分析过程中，对代码进行了一些理解，但是由于自己接触图像处理和机器视觉没多久，另外由于自己编程能力比较弱，所以分析过程可能会有不少的错误，希望各位不吝指正。而且，因为编程很多地方不懂，所以注释得非常乱，还海涵。

run\_tld.cpp

**[cpp]** [view plaincopy](http://blog.csdn.net/zouxy09/article/details/7893032)

1. #include <opencv2/opencv.hpp>
2. #include <tld\_utils.h>
3. #include <iostream>
4. #include <sstream>  //c++中的sstream类，提供了程序和string对象之间的I/O，可以通过ostringstream
5. //和instringstream两个类来声明对象，分别对应输出流和输入流
6. #include <TLD.h>
7. #include <stdio.h>
8. **using** **namespace** cv;
9. **using** **namespace** std;
10. //Global variables
11. Rect box;
12. **bool** drawing\_box = **false**;
13. **bool** gotBB = **false**;
14. **bool** tl = **true**;
15. **bool** rep = **false**;
16. **bool** fromfile=**false**;
17. string video;
19. //读取记录bounding box的文件，获得bounding box的四个参数：左上角坐标x，y和宽高
20. /\*如在\datasets\06\_car\init.txt中：记录了初始目标的bounding box，内容如下
21. 142,125,232,164
22. \*/
23. **void** readBB(**char**\* file){
24. ifstream bb\_file (file);  //以输入方式打开文件
25. string line;
26. //istream& getline ( istream& , string& );
27. //将输入流is中读到的字符存入str中，终结符默认为 '\n'（换行符）
28. getline(bb\_file, line);
29. istringstream linestream(line); //istringstream对象可以绑定一行字符串，然后以空格为分隔符把该行分隔开来。
30. string x1,y1,x2,y2;
32. //istream& getline ( istream &is , string &str , char delim );
33. //将输入流is中读到的字符存入str中，直到遇到终结符delim才结束。
34. getline (linestream,x1, ',');
35. getline (linestream,y1, ',');
36. getline (linestream,x2, ',');
37. getline (linestream,y2, ',');
39. //atoi 功 能： 把字符串转换成整型数
40. **int** x = atoi(x1.c\_str());// = (int)file["bb\_x"];
41. **int** y = atoi(y1.c\_str());// = (int)file["bb\_y"];
42. **int** w = atoi(x2.c\_str())-x;// = (int)file["bb\_w"];
43. **int** h = atoi(y2.c\_str())-y;// = (int)file["bb\_h"];
44. box = Rect(x,y,w,h);
45. }
47. //bounding box mouse callback
48. //鼠标的响应就是得到目标区域的范围，用鼠标选中bounding box。
49. **void** mouseHandler(**int** event, **int** x, **int** y, **int** flags, **void** \*param){
50. **switch**( event ){
51. **case** CV\_EVENT\_MOUSEMOVE:
52. **if** (drawing\_box){
53. box.width = x-box.x;
54. box.height = y-box.y;
55. }
56. **break**;
57. **case** CV\_EVENT\_LBUTTONDOWN:
58. drawing\_box = **true**;
59. box = Rect( x, y, 0, 0 );
60. **break**;
61. **case** CV\_EVENT\_LBUTTONUP:
62. drawing\_box = **false**;
63. **if**( box.width < 0 ){
64. box.x += box.width;
65. box.width \*= -1;
66. }
67. **if**( box.height < 0 ){
68. box.y += box.height;
69. box.height \*= -1;
70. }
71. gotBB = **true**;   //已经获得bounding box
72. **break**;
73. }
74. }
76. **void** print\_help(**char**\*\* argv){
77. printf("use:\n     %s -p /path/parameters.yml\n",argv[0]);
78. printf("-s    source video\n-b        bounding box file\n-tl  track and learn\n-r     repeat\n");
79. }
81. //分析运行程序时的命令行参数
82. **void** read\_options(**int** argc, **char**\*\* argv, VideoCapture& capture, FileStorage &fs){
83. **for** (**int** i=0;i<argc;i++){
84. **if** (strcmp(argv[i],"-b")==0){
85. **if** (argc>i){
86. readBB(argv[i+1]);  //是否指定初始的bounding box
87. gotBB = **true**;
88. }
89. **else**
90. print\_help(argv);
91. }
92. **if** (strcmp(argv[i],"-s")==0){   //从视频文件中读取
93. **if** (argc>i){
94. video = string(argv[i+1]);
95. capture.open(video);
96. fromfile = **true**;
97. }
98. **else**
99. print\_help(argv);
101. }
102. //Similar in format to XML, Yahoo! Markup Language (YML) provides functionality to Open
103. //Applications in a safe and standardized fashion. You include YML tags in the HTML code
104. //of an Open Application.
105. **if** (strcmp(argv[i],"-p")==0){   //读取参数文件parameters.yml
106. **if** (argc>i){
107. //FileStorage类的读取方式可以是：FileStorage fs(".\\parameters.yml", FileStorage::READ);
108. fs.open(argv[i+1], FileStorage::READ);
109. }
110. **else**
111. print\_help(argv);
112. }
113. **if** (strcmp(argv[i],"-no\_tl")==0){  //To train only in the first frame (no tracking, no learning)
114. tl = **false**;
115. }
116. **if** (strcmp(argv[i],"-r")==0){  //Repeat the video, first time learns, second time detects
117. rep = **true**;
118. }
119. }
120. }
122. /\*
123. 运行程序时：
124. %To run from camera
125. ./run\_tld -p ../parameters.yml
126. %To run from file
127. ./run\_tld -p ../parameters.yml -s ../datasets/06\_car/car.mpg
128. %To init bounding box from file
129. ./run\_tld -p ../parameters.yml -s ../datasets/06\_car/car.mpg -b ../datasets/06\_car/init.txt
130. %To train only in the first frame (no tracking, no learning)
131. ./run\_tld -p ../parameters.yml -s ../datasets/06\_car/car.mpg -b ../datasets/06\_car/init.txt -no\_tl
132. %To test the final detector (Repeat the video, first time learns, second time detects)
133. ./run\_tld -p ../parameters.yml -s ../datasets/06\_car/car.mpg -b ../datasets/06\_car/init.txt -r
134. \*/
135. //感觉就是对起始帧进行初始化工作，然后逐帧读入图片序列，进行算法处理。
136. **int** main(**int** argc, **char** \* argv[]){
137. VideoCapture capture;
138. capture.open(0);
140. //OpenCV的C++接口中，用于保存图像的imwrite只能保存整数数据，且需作为图像格式。当需要保存浮
141. //点数据或XML/YML文件时，OpenCV的C语言接口提供了cvSave函数，但这一函数在C++接口中已经被删除。
142. //取而代之的是FileStorage类。
143. FileStorage fs;
144. //Read options
145. read\_options(argc, argv, capture, fs);  //分析命令行参数
146. //Init camera
147. **if** (!capture.isOpened())
148. {
149. cout << "capture device failed to open!" << endl;
150. **return** 1;
151. }
152. //Register mouse callback to draw the bounding box
153. cvNamedWindow("TLD",CV\_WINDOW\_AUTOSIZE);
154. cvSetMouseCallback( "TLD", mouseHandler, NULL );  //用鼠标选中初始目标的bounding box
155. //TLD framework
156. TLD tld;
157. //Read parameters file
158. tld.read(fs.getFirstTopLevelNode());
159. Mat frame;
160. Mat last\_gray;
161. Mat first;
162. **if** (fromfile){  //如果指定为从文件读取
163. capture >> frame;   //读当前帧
164. cvtColor(frame, last\_gray, CV\_RGB2GRAY);  //转换为灰度图像
165. frame.copyTo(first);  //拷贝作为第一帧
166. }**else**{   //如果为读取摄像头，则设置获取的图像大小为320x240
167. capture.set(CV\_CAP\_PROP\_FRAME\_WIDTH,340);  //340？？
168. capture.set(CV\_CAP\_PROP\_FRAME\_HEIGHT,240);
169. }
171. ///Initialization
172. GETBOUNDINGBOX:   //标号：获取bounding box
173. **while**(!gotBB)
174. {
175. **if** (!fromfile){
176. capture >> frame;
177. }
178. **else**
179. first.copyTo(frame);
180. cvtColor(frame, last\_gray, CV\_RGB2GRAY);
181. drawBox(frame,box);  //把bounding box 画出来
182. imshow("TLD", frame);
183. **if** (cvWaitKey(33) == 'q')
184. **return** 0;
185. }
186. //由于图像片（min\_win 为15x15像素）是在bounding box中采样得到的，所以box必须比min\_win要大
187. **if** (min(box.width, box.height)<(**int**)fs.getFirstTopLevelNode()["min\_win"]){
188. cout << "Bounding box too small, try again." << endl;
189. gotBB = **false**;
190. **goto** GETBOUNDINGBOX;
191. }
192. //Remove callback
193. cvSetMouseCallback( "TLD", NULL, NULL );  //如果已经获得第一帧用户框定的box了，就取消鼠标响应
194. printf("Initial Bounding Box = x:%d y:%d h:%d w:%d\n",box.x,box.y,box.width,box.height);
195. //Output file
196. **FILE**  \*bb\_file = fopen("bounding\_boxes.txt","w");
198. //TLD initialization
199. tld.init(last\_gray, box, bb\_file);
201. ///Run-time
202. Mat current\_gray;
203. BoundingBox pbox;
204. vector<Point2f> pts1;
205. vector<Point2f> pts2;
206. **bool** status=**true**;  //记录跟踪成功与否的状态 lastbox been found
207. **int** frames = 1;  //记录已过去帧数
208. **int** detections = 1;  //记录成功检测到的目标box数目
210. REPEAT:
211. **while**(capture.read(frame)){
212. //get frame
213. cvtColor(frame, current\_gray, CV\_RGB2GRAY);
214. //Process Frame
215. tld.processFrame(last\_gray, current\_gray, pts1, pts2, pbox, status, tl, bb\_file);
216. //Draw Points
217. **if** (status){  //如果跟踪成功
218. drawPoints(frame,pts1);
219. drawPoints(frame,pts2,Scalar(0,255,0));  //当前的特征点用蓝色点表示
220. drawBox(frame,pbox);
221. detections++;
222. }
223. //Display
224. imshow("TLD", frame);
225. //swap points and images
226. swap(last\_gray, current\_gray);  //STL函数swap()用来交换两对象的值。其泛型化版本定义于<algorithm>;
227. pts1.clear();
228. pts2.clear();
229. frames++;
230. printf("Detection rate: %d/%d\n", detections, frames);
231. **if** (cvWaitKey(33) == 'q')
232. **break**;
233. }
234. **if** (rep){
235. rep = **false**;
236. tl = **false**;
237. fclose(bb\_file);
238. bb\_file = fopen("final\_detector.txt","w");
239. //capture.set(CV\_CAP\_PROP\_POS\_AVI\_RATIO,0);
240. capture.release();
241. capture.open(video);
242. **goto** REPEAT;
243. }
244. fclose(bb\_file);
245. **return** 0;
246. }

tld\_utils.cpp

**[cpp]** [view plaincopy](http://blog.csdn.net/zouxy09/article/details/7893032)

1. #include <tld\_utils.h>
2. **using** **namespace** cv;
3. **using** **namespace** std;
5. /\*vector是C++标准模板库STL中的部分内容，它是一个多功能的，能够操作多种数据结构和算法的
6. 模板类和函数库。vector之所以被认为是一个容器，是因为它能够像容器一样存放各种类型的对象，
7. 简单地说，vector是一个能够存放任意类型的动态数组，能够增加和压缩数据。
8. 为了可以使用vector，必须在你的头文件中包含下面的代码：
9. #include <vector>
10. vector属于std命名域的，因此需要通过命名限定，如下完成你的代码：
11. using std::vector;
12. \*/
14. **void** drawBox(Mat& image, CvRect box, Scalar color, **int** thick){
15. rectangle( image, cvPoint(box.x, box.y), cvPoint(box.x+box.width,box.y+box.height),color, thick);
16. }
18. //函数 cvRound, cvFloor, cvCeil 用一种舍入方法将输入浮点数转换成整数。
19. //cvRound 返回和参数最接近的整数值。 cvFloor 返回不大于参数的最大整数值。
20. //cvCeil 返回不小于参数的最小整数值。
21. **void** drawPoints(Mat& image, vector<Point2f> points,Scalar color){
22. **for**( vector<Point2f>::const\_iterator i = points.begin(), ie = points.end(); i != ie; ++i )
23. {
24. Point center( cvRound(i->x ), cvRound(i->y));  //类似于int i(3)的初始化，但center为何没用到？
25. circle(image,\*i,2,color,1);
26. }
27. }
29. Mat createMask(**const** Mat& image, CvRect box){
30. Mat mask = Mat::zeros(image.rows,image.cols,CV\_8U);
31. drawBox(mask,box,Scalar::all(255),CV\_FILLED);
32. **return** mask;
33. }
35. //STL中的nth\_element()方法找出一个数列中排名第n的那个数。
36. //对于序列a[0:len-1]将第n大的数字，排在a[n],同时a[0:n-1]都小于a[n],a[n+1:]都大于a[n],
37. //但a[n]左右的这两个序列不一定有序。
38. //用在中值流跟踪算法中，寻找中值
39. **float** median(vector<**float**> v)
40. {
41. **int** n = floor(v.size() / 2);
42. nth\_element(v.begin(), v.begin()+n, v.end());
43. **return** v[n];
44. }
46. //<algorithm> //random\_shuffle的头文件
47. //shuffle 洗牌  首先简单的介绍一个扑克牌洗牌的方法，假设一个数组 poker[52] 中存有一副扑克
48. //牌1-52的牌点值，使用一个for循环遍历这个数组，每次循环都生成一个[0，52)之间的随机数RandNum，
49. //以RandNum为数组下标，把当前下标对应的值和RandNum对应位置的值交换，循环结束，每个牌都与某个
50. //位置交换了一次，这样一副牌就被打乱了。 理解代码如下：
51. /\*
52. for (int i = 0; i < 52; ++i)
53. {
54. int RandNum = rand() % 52;
55. int tmp = poker[i];
56. poker[i] = poker[RandNum];
57. poker[RandNum] = tmp;
58. }
59. \*/
60. //需要指定范围内的随机数，传统的方法是使用ANSI C的函数random(),然后格式化结果以便结果是落在
61. //指定的范围内。但是，使用这个方法至少有两个缺点。做格式化时，结果常常是扭曲的,且只支持整型数。
62. //C++中提供了更好的解决方法，那就是STL中的random\_shuffle()算法。产生指定范围内的随机元素集的最佳方法
63. //是创建一个顺序序列（也就是向量或者内置数组），在这个顺序序列中含有指定范围的所有值。
64. //例如，如果你需要产生100个0-99之间的数，那么就创建一个向量并用100个按升序排列的数填充向量.
65. //填充完向量之后，用random\_shuffle()算法打乱元素排列顺序。
66. //默认的random\_shuffle中, 被操作序列的index 与 rand() % N 两个位置的值交换，来达到乱序的目的。
67. //index\_shuffle()用于产生指定范围[begin:end]的随机数，返回随机数数组
68. vector<**int**> index\_shuffle(**int** begin,**int** end){
69. vector<**int**> indexes(end-begin);
70. **for** (**int** i=begin;i<end;i++){
71. indexes[i]=i;
72. }
73. random\_shuffle(indexes.begin(),indexes.end());
74. **return** indexes;
75. }

## [LKTracker](http://blog.csdn.net/zouxy09/article/details/7893056)

       下面是自己在看论文和这些大牛的分析过程中，对代码进行了一些理解，但是由于自己接触图像处理和机器视觉没多久，另外由于自己编程能力比较弱，所以分析过程可能会有不少的错误，希望各位不吝指正。而且，因为编程很多地方不懂，所以注释得非常乱，还海涵。

LKTracker.h

**[cpp]** [view plaincopy](http://blog.csdn.net/zouxy09/article/details/7893056)

1. #include<tld\_utils.h>
2. #include <opencv2/opencv.hpp>
4. //使用金字塔LK光流法跟踪，所以类的成员变量很多都是OpenCV中calcOpticalFlowPyrLK()函数的参数
5. **class** LKTracker{
6. **private**:
7. std::vector<cv::Point2f> pointsFB;
8. cv::Size window\_size;  //每个金字塔层的搜索窗口尺寸
9. **int** level;            //最大的金字塔层数
10. std::vector<uchar> status;   //数组。如果对应特征的光流被发现，数组中的每一个元素都被设置为 1， 否则设置为 0
11. std::vector<uchar> FB\_status;
12. std::vector<**float**> similarity;  //相似度
13. std::vector<**float**> FB\_error;   //Forward-Backward error方法，求FB\_error的结果与原始位置的欧式距离
14. //做比较，把距离过大的跟踪结果舍弃
15. **float** simmed;
16. **float** fbmed;
17. //TermCriteria模板类，取代了之前的CvTermCriteria，这个类是作为迭代算法的终止条件的
18. //该类变量需要3个参数，一个是类型，第二个参数为迭代的最大次数，最后一个是特定的阈值。
19. //指定在每个金字塔层，为某点寻找光流的迭代过程的终止条件。
20. cv::TermCriteria term\_criteria;
21. **float** lambda;   //某阈值？？Lagrangian 乘子
22. // NCC 归一化交叉相关，FB error与NCC结合，使跟踪更稳定  交叉相关的图像匹配算法？？
23. //交叉相关法的作用是进行云团移动的短时预测。选取连续两个时次的GMS-5卫星云图，将云图区域划分为32×32像素
24. //的图像子集，采用交叉相关法计算获取两幅云图的最佳匹配区域，根据前后云图匹配区域的位置和时间间隔，确
25. //定出每个图像子集的移动矢量（速度和方向），并对图像子集的移动矢量进行客观分析，其后，基于检验后的云
26. //图移动矢量集，利用后向轨迹方法对云图作短时外推预测。
27. **void** normCrossCorrelation(**const** cv::Mat& img1, **const** cv::Mat& img2, std::vector<cv::Point2f>& points1, std::vector<cv::Point2f>& points2);
28. **bool** filterPts(std::vector<cv::Point2f>& points1,std::vector<cv::Point2f>& points2);
29. **public**:
30. LKTracker();
31. //特征点的跟踪？？
32. **bool** trackf2f(**const** cv::Mat& img1, **const** cv::Mat& img2,
33. std::vector<cv::Point2f> &points1, std::vector<cv::Point2f> &points2);
34. **float** getFB(){**return** fbmed;}
35. };

LKTracker.cpp

**[cpp]** [view plaincopy](http://blog.csdn.net/zouxy09/article/details/7893056)

1. #include <LKTracker.h>
2. **using** **namespace** cv;
4. //金字塔LK光流法跟踪
5. //Media Flow 中值光流跟踪 加 跟踪错误检测
6. //构造函数，初始化成员变量
7. LKTracker::LKTracker(){
8. ////该类变量需要3个参数，一个是类型，第二个参数为迭代的最大次数，最后一个是特定的阈值。
9. term\_criteria = TermCriteria( TermCriteria::COUNT + TermCriteria::EPS, 20, 0.03);
10. window\_size = Size(4,4);
11. level = 5;
12. lambda = 0.5;
13. }

16. **bool** LKTracker::trackf2f(**const** Mat& img1, **const** Mat& img2, vector<Point2f> &points1, vector<cv::Point2f> &points2){
17. //TODO!:implement c function cvCalcOpticalFlowPyrLK() or Faster tracking function
18. //Forward-Backward tracking
19. //基于Forward-Backward Error的中值流跟踪方法
20. //金字塔LK光流法跟踪
21. //forward trajectory 前向轨迹跟踪
22. calcOpticalFlowPyrLK( img1,img2, points1, points2, status, similarity, window\_size, level, term\_criteria, lambda, 0);
23. //backward trajectory 后向轨迹跟踪
24. calcOpticalFlowPyrLK( img2,img1, points2, pointsFB, FB\_status,FB\_error, window\_size, level, term\_criteria, lambda, 0);
26. //Compute the real FB-error
27. //原理很简单：从t时刻的图像的A点，跟踪到t+1时刻的图像B点；然后倒回来，从t+1时刻的图像的B点往回跟踪，
28. //假如跟踪到t时刻的图像的C点，这样就产生了前向和后向两个轨迹，比较t时刻中 A点 和 C点 的距离，如果距离
29. //小于一个阈值，那么就认为前向跟踪是正确的；这个距离就是FB\_error
30. //计算 前向 与 后向 轨迹的误差
31. **for**( **int** i= 0; i<points1.size(); ++i ){
32. FB\_error[i] = norm(pointsFB[i]-points1[i]);   //norm()求矩阵或向量的范数??绝对值？
33. }
34. //Filter out points with FB\_error[i] <= median(FB\_error) && points with sim\_error[i] > median(sim\_error)
35. normCrossCorrelation(img1, img2, points1, points2);
36. **return** filterPts(points1, points2);
37. }
39. //利用NCC把跟踪预测的结果周围取10\*10的小图片与原始位置周围10\*10的小图片（使用函数getRectSubPix得到）进
40. //行模板匹配（调用matchTemplate）
41. **void** LKTracker::normCrossCorrelation(**const** Mat& img1,**const** Mat& img2, vector<Point2f>& points1, vector<Point2f>& points2) {
42. Mat rec0(10,10,CV\_8U);
43. Mat rec1(10,10,CV\_8U);
44. Mat res(1,1,CV\_32F);
46. **for** (**int** i = 0; i < points1.size(); i++) {
47. **if** (status[i] == 1) {  //为1表示该特征点跟踪成功
48. //从前一帧和当前帧图像中（以每个特征点为中心？）提取10x10象素矩形，使用亚象素精度
49. getRectSubPix( img1, Size(10,10), points1[i],rec0 );
50. getRectSubPix( img2, Size(10,10), points2[i],rec1);
51. //匹配前一帧和当前帧中提取的10x10象素矩形，得到匹配后的映射图像
52. //CV\_TM\_CCOEFF\_NORMED 归一化相关系数匹配法
53. //参数分别为：欲搜索的图像。搜索模板。比较结果的映射图像。指定匹配方法
54. matchTemplate( rec0,rec1, res, CV\_TM\_CCOEFF\_NORMED);
55. similarity[i] = ((**float** \*)(res.data))[0];  //得到各个特征点的相似度大小
57. } **else** {
58. similarity[i] = 0.0;
59. }
60. }
61. rec0.release();
62. rec1.release();
63. res.release();
64. }
66. //筛选出 FB\_error[i] <= median(FB\_error) 和 sim\_error[i] > median(sim\_error) 的特征点
67. //得到NCC和FB error结果的中值，分别去掉中值一半的跟踪结果不好的点
68. **bool** LKTracker::filterPts(vector<Point2f>& points1,vector<Point2f>& points2){
69. //Get Error Medians
70. simmed = median(similarity);   //找到相似度的中值
71. **size\_t** i, k;
72. **for**( i=k = 0; i<points2.size(); ++i ){
73. **if**( !status[i])
74. **continue**;
75. **if**(similarity[i]> simmed){   //剩下 similarity[i]> simmed 的特征点
76. points1[k] = points1[i];
77. points2[k] = points2[i];
78. FB\_error[k] = FB\_error[i];
79. k++;
80. }
81. }
82. **if** (k==0)
83. **return** **false**;
84. points1.resize(k);
85. points2.resize(k);
86. FB\_error.resize(k);
88. fbmed = median(FB\_error);     //找到FB\_error的中值
89. **for**( i=k = 0; i<points2.size(); ++i ){
90. **if**( !status[i])
91. **continue**;
92. **if**(FB\_error[i] <= fbmed){   /
93. points1[k] = points1[i];   //再对上一步剩下的特征点进一步筛选，剩下 FB\_error[i] <= fbmed 的特征点
94. points2[k] = points2[i];
95. k++;
96. }
97. }
98. points1.resize(k);
99. points2.resize(k);
100. **if** (k>0)
101. **return** **true**;
102. **else**
103. **return** **false**;
104. }
105. /\*
106. \* old OpenCV style
107. void LKTracker::init(Mat img0, vector<Point2f> &points){
108. //Preallocate
109. //pyr1 = cvCreateImage(Size(img1.width+8,img1.height/3),IPL\_DEPTH\_32F,1);   //pyr2 = cvCreateImage(Size(img1.width+8,img1.height/3),IPL\_DEPTH\_32F,1);   //const int NUM\_PTS = points.size();
110. //status = new char[NUM\_PTS];
111. //track\_error = new float[NUM\_PTS];
112. //FB\_error = new float[NUM\_PTS];
113. }
115. void LKTracker::trackf2f(..){
116. cvCalcOpticalFlowPyrLK( &img1, &img2, pyr1, pyr1, points1, points2, points1.size(), window\_size, level, status, track\_error, term\_criteria, CV\_LKFLOW\_INITIAL\_GUESSES);
117. cvCalcOpticalFlowPyrLK( &img2, &img1, pyr2, pyr1, points2, pointsFB, points2.size(),window\_size, level, 0, 0, term\_criteria, CV\_LKFLOW\_INITIAL\_GUESSES | CV\_LKFLOW\_PYR\_A\_READY | CV\_LKFLOW\_PYR\_B\_READY );
118. }
119. \*/

## TLD.h

      下面是自己在看论文和这些大牛的分析过程中，对代码进行了一些理解，但是由于自己接触图像处理和机器视觉没多久，另外由于自己编程能力比较弱，所以分析过程可能会有不少的错误，希望各位不吝指正。而且，因为编程很多地方不懂，所以注释得非常乱，还海涵。

TLD.h

**[cpp]** [view plaincopy](http://blog.csdn.net/zouxy09/article/details/7893081)

1. #include <opencv2/opencv.hpp>
2. #include <tld\_utils.h>
3. #include <LKTracker.h>
4. #include <FerNNClassifier.h>
5. #include <fstream>

8. //Bounding Boxes
9. **struct** BoundingBox : **public** cv::Rect {
10. BoundingBox(){}
11. BoundingBox(cv::Rect r): cv::Rect(r){}   //继承的话需要初始化基类
12. **public**:
13. **float** overlap;        //Overlap with current Bounding Box
14. **int** sidx;             //scale index
15. };
17. //Detection structure
18. **struct** DetStruct {
19. std::vector<**int**> bb;
20. std::vector<std::vector<**int**> > patt;
21. std::vector<**float**> conf1;
22. std::vector<**float**> conf2;
23. std::vector<std::vector<**int**> > isin;
24. std::vector<cv::Mat> patch;
25. };
27. //Temporal structure
28. **struct** TempStruct {
29. std::vector<std::vector<**int**> > patt;
30. std::vector<**float**> conf;
31. };
33. **struct** OComparator{  //比较两者重合度
34. OComparator(**const** std::vector<BoundingBox>& \_grid):grid(\_grid){}
35. std::vector<BoundingBox> grid;
36. **bool** operator()(**int** idx1,**int** idx2){
37. **return** grid[idx1].overlap > grid[idx2].overlap;
38. }
39. };
41. **struct** CComparator{  //比较两者确信度？
42. CComparator(**const** std::vector<**float**>& \_conf):conf(\_conf){}
43. std::vector<**float**> conf;
44. **bool** operator()(**int** idx1,**int** idx2){
45. **return** conf[idx1]> conf[idx2];
46. }
47. };

50. **class** TLD{
51. **private**:
52. cv::PatchGenerator generator;  //PatchGenerator类用来对图像区域进行仿射变换
53. FerNNClassifier classifier;
54. LKTracker tracker;
56. //下面这些参数通过程序开始运行时读入parameters.yml文件进行初始化
57. ///Parameters
58. **int** bbox\_step;
59. **int** min\_win;
60. **int** patch\_size;
62. //initial parameters for positive examples
63. //从第一帧得到的目标的bounding box中（文件读取或者用户框定），经过几何变换得
64. //到 num\_closest\_init \* num\_warps\_init 个正样本
65. **int** num\_closest\_init;  //最近邻窗口数 10
66. **int** num\_warps\_init;  //几何变换数目 20
67. **int** noise\_init;
68. **float** angle\_init;
69. **float** shift\_init;
70. **float** scale\_init;
72. ////从跟踪得到的目标的bounding box中，经过几何变换更新正样本（添加到在线模型？）
73. //update parameters for positive examples
74. **int** num\_closest\_update;
75. **int** num\_warps\_update;
76. **int** noise\_update;
77. **float** angle\_update;
78. **float** shift\_update;
79. **float** scale\_update;
81. //parameters for negative examples
82. **float** bad\_overlap;
83. **float** bad\_patches;
85. ///Variables
86. //Integral Images  积分图像，用以计算2bitBP特征（类似于haar特征的计算）
87. //Mat最大的优势跟STL很相似，都是对内存进行动态的管理，不需要之前用户手动的管理内存
88. cv::Mat iisum;
89. cv::Mat iisqsum;
90. **float** var;
92. //Training data
93. //std::pair主要的作用是将两个数据组合成一个数据，两个数据可以是同一类型或者不同类型。
94. //pair实质上是一个结构体，其主要的两个成员变量是first和second，这两个变量可以直接使用。
95. //在这里用来表示样本，first成员为 features 特征点数组，second成员为 labels 样本类别标签
96. std::vector<std::pair<std::vector<**int**>,**int**> > pX; //positive ferns <features,labels=1>  正样本
97. std::vector<std::pair<std::vector<**int**>,**int**> > nX; // negative ferns <features,labels=0>  负样本
98. cv::Mat pEx;  //positive NN example
99. std::vector<cv::Mat> nEx; //negative NN examples
101. //Test data
102. std::vector<std::pair<std::vector<**int**>,**int**> > nXT; //negative data to Test
103. std::vector<cv::Mat> nExT; //negative NN examples to Test
105. //Last frame data
106. BoundingBox lastbox;
107. **bool** lastvalid;
108. **float** lastconf;
110. //Current frame data
111. //Tracker data
112. **bool** tracked;
113. BoundingBox tbb;
114. **bool** tvalid;
115. **float** tconf;
117. //Detector data
118. TempStruct tmp;
119. DetStruct dt;
120. std::vector<BoundingBox> dbb;
121. std::vector<**bool**> dvalid;   //检测有效性？？
122. std::vector<**float**> dconf;  //检测确信度？？
123. **bool** detected;

126. //Bounding Boxes
127. std::vector<BoundingBox> grid;
128. std::vector<cv::Size> scales;
129. std::vector<**int**> good\_boxes; //indexes of bboxes with overlap > 0.6
130. std::vector<**int**> bad\_boxes; //indexes of bboxes with overlap < 0.2
131. BoundingBox bbhull; // hull of good\_boxes  //good\_boxes 的 壳，也就是窗口的边框
132. BoundingBox best\_box; // maximum overlapping bbox
134. **public**:
135. //Constructors
136. TLD();
137. TLD(**const** cv::FileNode& file);
138. **void** read(**const** cv::FileNode& file);
140. //Methods
141. **void** init(**const** cv::Mat& frame1,**const** cv::Rect &box, **FILE**\* bb\_file);
142. **void** generatePositiveData(**const** cv::Mat& frame, **int** num\_warps);
143. **void** generateNegativeData(**const** cv::Mat& frame);
144. **void** processFrame(**const** cv::Mat& img1,**const** cv::Mat& img2,std::vector<cv::Point2f>& points1,std::vector<cv::Point2f>& points2,
145. BoundingBox& bbnext,**bool**& lastboxfound, **bool** tl,**FILE**\* bb\_file);
146. **void** track(**const** cv::Mat& img1, **const** cv::Mat& img2,std::vector<cv::Point2f>& points1,std::vector<cv::Point2f>& points2);
147. **void** detect(**const** cv::Mat& frame);
148. **void** clusterConf(**const** std::vector<BoundingBox>& dbb,**const** std::vector<**float**>& dconf,std::vector<BoundingBox>& cbb,std::vector<**float**>& cconf);
149. **void** evaluate();
150. **void** learn(**const** cv::Mat& img);
152. //Tools
153. **void** buildGrid(**const** cv::Mat& img, **const** cv::Rect& box);
154. **float** bbOverlap(**const** BoundingBox& box1,**const** BoundingBox& box2);
155. **void** getOverlappingBoxes(**const** cv::Rect& box1,**int** num\_closest);
156. **void** getBBHull();
157. **void** getPattern(**const** cv::Mat& img, cv::Mat& pattern,cv::Scalar& mean,cv::Scalar& stdev);
158. **void** bbPoints(std::vector<cv::Point2f>& points, **const** BoundingBox& bb);
159. **void** bbPredict(**const** std::vector<cv::Point2f>& points1,**const** std::vector<cv::Point2f>& points2,
160. **const** BoundingBox& bb1,BoundingBox& bb2);
161. **double** getVar(**const** BoundingBox& box,**const** cv::Mat& sum,**const** cv::Mat& sqsum);
162. **bool** bbComp(**const** BoundingBox& bb1,**const** BoundingBox& bb2);
163. **int** clusterBB(**const** std::vector<BoundingBox>& dbb,std::vector<**int**>& indexes);
164. };

TLD.cpp

**[cpp]** [view plaincopy](http://blog.csdn.net/zouxy09/article/details/7893081)

1. /\*
2. \* TLD.cpp
3. \*
4. \*  Created on: Jun 9, 2011
5. \*      Author: alantrrs
6. \*/
8. #include <TLD.h>
9. #include <stdio.h>
10. **using** **namespace** cv;
11. **using** **namespace** std;

14. TLD::TLD()
15. {
16. }
17. TLD::TLD(**const** FileNode& file){
18. read(file);
19. }
21. **void** TLD::read(**const** FileNode& file){
22. ///Bounding Box Parameters
23. min\_win = (**int**)file["min\_win"];
24. ///Genarator Parameters
25. //initial parameters for positive examples
26. patch\_size = (**int**)file["patch\_size"];
27. num\_closest\_init = (**int**)file["num\_closest\_init"];
28. num\_warps\_init = (**int**)file["num\_warps\_init"];
29. noise\_init = (**int**)file["noise\_init"];
30. angle\_init = (**float**)file["angle\_init"];
31. shift\_init = (**float**)file["shift\_init"];
32. scale\_init = (**float**)file["scale\_init"];
33. //update parameters for positive examples
34. num\_closest\_update = (**int**)file["num\_closest\_update"];
35. num\_warps\_update = (**int**)file["num\_warps\_update"];
36. noise\_update = (**int**)file["noise\_update"];
37. angle\_update = (**float**)file["angle\_update"];
38. shift\_update = (**float**)file["shift\_update"];
39. scale\_update = (**float**)file["scale\_update"];
40. //parameters for negative examples
41. bad\_overlap = (**float**)file["overlap"];
42. bad\_patches = (**int**)file["num\_patches"];
43. classifier.read(file);
44. }
46. //此函数完成准备工作
47. **void** TLD::init(**const** Mat& frame1, **const** Rect& box, **FILE**\* bb\_file){
48. //bb\_file = fopen("bounding\_boxes.txt","w");
49. //Get Bounding Boxes
50. //此函数根据传入的box（目标边界框）在传入的图像frame1中构建全部的扫描窗口，并计算重叠度
51. buildGrid(frame1, box);
52. printf("Created %d bounding boxes\n",(**int**)grid.size());  //vector的成员size()用于获取向量元素的个数
54. ///Preparation
55. //allocation
56. //积分图像，用以计算2bitBP特征（类似于haar特征的计算）
57. //Mat的创建，方式有两种：1.调用create（行，列，类型）2.Mat（行，列，类型（值））。
58. iisum.create(frame1.rows+1, frame1.cols+1, CV\_32F);
59. iisqsum.create(frame1.rows+1, frame1.cols+1, CV\_64F);
61. //Detector data中定义：std::vector<float> dconf;  检测确信度？？
62. //vector 的reserve增加了vector的capacity，但是它的size没有改变！而resize改变了vector
63. //的capacity同时也增加了它的size！reserve是容器预留空间，但在空间内不真正创建元素对象，
64. //所以在没有添加新的对象之前，不能引用容器内的元素。
65. //不管是调用resize还是reserve，二者对容器原有的元素都没有影响。
66. //myVec.reserve( 100 );     // 新元素还没有构造, 此时不能用[]访问元素
67. //myVec.resize( 100 );      // 用元素的默认构造函数构造了100个新的元素，可以直接操作新元素
68. dconf.reserve(100);
69. dbb.reserve(100);
70. bbox\_step =7;
72. //以下在Detector data中定义的容器都给其分配grid.size()大小（这个是一幅图像中全部的扫描窗口个数）的容量
73. //Detector data中定义TempStruct tmp;
74. //tmp.conf.reserve(grid.size());
75. tmp.conf = vector<**float**>(grid.size());
76. tmp.patt = vector<vector<**int**> >(grid.size(), vector<**int**>(10,0));
77. //tmp.patt.reserve(grid.size());
78. dt.bb.reserve(grid.size());
79. good\_boxes.reserve(grid.size());
80. bad\_boxes.reserve(grid.size());
82. //TLD中定义：cv::Mat pEx;  //positive NN example 大小为15\*15图像片
83. pEx.create(patch\_size, patch\_size, CV\_64F);
85. //Init Generator
86. //TLD中定义：cv::PatchGenerator generator;  //PatchGenerator类用来对图像区域进行仿射变换
87. /\*
88. cv::PatchGenerator::PatchGenerator (
89. double     \_backgroundMin,
90. double     \_backgroundMax,
91. double     \_noiseRange,
92. bool     \_randomBlur = true,
93. double     \_lambdaMin = 0.6,
94. double     \_lambdaMax = 1.5,
95. double     \_thetaMin = -CV\_PI,
96. double     \_thetaMax = CV\_PI,
97. double     \_phiMin = -CV\_PI,
98. double     \_phiMax = CV\_PI
99. )
100. 一般的用法是先初始化一个PatchGenerator的实例，然后RNG一个随机因子，再调用（）运算符产生一个变换后的正样本。
101. \*/
102. generator = PatchGenerator (0,0,noise\_init,**true**,1-scale\_init,1+scale\_init,-angle\_init\*CV\_PI/180,
103. angle\_init\*CV\_PI/180,-angle\_init\*CV\_PI/180,angle\_init\*CV\_PI/180);
105. //此函数根据传入的box（目标边界框），在整帧图像中的全部窗口中寻找与该box距离最小（即最相似，
106. //重叠度最大）的num\_closest\_init个窗口，然后把这些窗口 归入good\_boxes容器
107. //同时，把重叠度小于0.2的，归入 bad\_boxes 容器
108. //首先根据overlap的比例信息选出重复区域比例大于60%并且前num\_closet\_init= 10个的最接近box的RectBox，
109. //相当于对RectBox进行筛选。并通过BBhull函数得到这些RectBox的最大边界。
110. getOverlappingBoxes(box, num\_closest\_init);
111. printf("Found %d good boxes, %d bad boxes\n",(**int**)good\_boxes.size(),(**int**)bad\_boxes.size());
112. printf("Best Box: %d %d %d %d\n",best\_box.x, best\_box.y, best\_box.width, best\_box.height);
113. printf("Bounding box hull: %d %d %d %d\n", bbhull.x, bbhull.y, bbhull.width, bbhull.height);
115. //Correct Bounding Box
116. lastbox=best\_box;
117. lastconf=1;
118. lastvalid=**true**;
119. //Print
120. fprintf(bb\_file,"%d,%d,%d,%d,%f\n",lastbox.x,lastbox.y,lastbox.br().x,lastbox.br().y,lastconf);
122. //Prepare Classifier 准备分类器
123. //scales容器里是所有扫描窗口的尺度，由buildGrid()函数初始化
124. classifier.prepare(scales);
126. ///Generate Data
127. // Generate positive data
128. generatePositiveData(frame1, num\_warps\_init);
130. // Set variance threshold
131. Scalar stdev, mean;
132. //统计best\_box的均值和标准差
133. ////例如需要提取图像A的某个ROI（感兴趣区域，由矩形框）的话，用Mat类的B=img(ROI)即可提取
134. //frame1(best\_box)就表示在frame1中提取best\_box区域（目标区域）的图像片
135. meanStdDev(frame1(best\_box), mean, stdev);
137. //利用积分图像去计算每个待检测窗口的方差
138. //cvIntegral( const CvArr\* image, CvArr\* sum, CvArr\* sqsum=NULL, CvArr\* tilted\_sum=NULL );
139. //计算积分图像，输入图像，sum积分图像, W+1×H+1，sqsum对象素值平方的积分图像，tilted\_sum旋转45度的积分图像
140. //利用积分图像，可以计算在某象素的上－右方的或者旋转的矩形区域中进行求和、求均值以及标准方差的计算，
141. //并且保证运算的复杂度为O(1)。
142. integral(frame1, iisum, iisqsum);
143. //级联分类器模块一：方差检测模块，利用积分图计算每个待检测窗口的方差，方差大于var阈值（目标patch方差的50%）的，
144. //则认为其含有前景目标方差；var 为标准差的平方
145. var = pow(stdev.val[0],2) \* 0.5; //getVar(best\_box,iisum,iisqsum);
146. cout << "variance: " << var << endl;
148. //check variance
149. //getVar函数通过积分图像计算输入的best\_box的方差
150. **double** vr =  getVar(best\_box, iisum, iisqsum)\*0.5;
151. cout << "check variance: " << vr << endl;
153. // Generate negative data
154. generateNegativeData(frame1);
156. //Split Negative Ferns into Training and Testing sets (they are already shuffled)
157. //将负样本放进 训练和测试集
158. **int** half = (**int**)nX.size()\*0.5f;
159. //vector::assign函数将区间[start, end)中的值赋值给当前的vector.
160. //将一半的负样本集 作为 测试集
161. nXT.assign(nX.begin()+half, nX.end());  //nXT; //negative data to Test
162. //然后将剩下的一半作为训练集
163. nX.resize(half);
165. ///Split Negative NN Examples into Training and Testing sets
166. half = (**int**)nEx.size()\*0.5f;
167. nExT.assign(nEx.begin()+half,nEx.end());
168. nEx.resize(half);
170. //Merge Negative Data with Positive Data and shuffle it
171. //将负样本和正样本合并，然后打乱
172. vector<pair<vector<**int**>,**int**> > ferns\_data(nX.size()+pX.size());
173. vector<**int**> idx = index\_shuffle(0, ferns\_data.size());
174. **int** a=0;
175. **for** (**int** i=0;i<pX.size();i++){
176. ferns\_data[idx[a]] = pX[i];
177. a++;
178. }
179. **for** (**int** i=0;i<nX.size();i++){
180. ferns\_data[idx[a]] = nX[i];
181. a++;
182. }
184. //Data already have been shuffled, just putting it in the same vector
185. vector<cv::Mat> nn\_data(nEx.size()+1);
186. nn\_data[0] = pEx;
187. **for** (**int** i=0;i<nEx.size();i++){
188. nn\_data[i+1]= nEx[i];
189. }
191. ///Training
192. //训练 集合分类器（森林） 和 最近邻分类器
193. classifier.trainF(ferns\_data, 2); //bootstrap = 2
194. classifier.trainNN(nn\_data);
196. ///Threshold Evaluation on testing sets
197. //用样本在上面得到的 集合分类器（森林） 和 最近邻分类器 中分类，评价得到最好的阈值
198. classifier.evaluateTh(nXT, nExT);
199. }
201. /\* Generate Positive data
202. \* Inputs:
203. \* - good\_boxes (bbP)
204. \* - best\_box (bbP0)
205. \* - frame (im0)
206. \* Outputs:
207. \* - Positive fern features (pX)
208. \* - Positive NN examples (pEx)
209. \*/
210. **void** TLD::generatePositiveData(**const** Mat& frame, **int** num\_warps){
211. /\*
212. CvScalar定义可存放1—4个数值的数值，常用来存储像素，其结构体如下：
213. typedef struct CvScalar
214. {
215. double val[4];
216. }CvScalar;
217. 如果使用的图像是1通道的，则s.val[0]中存储数据
218. 如果使用的图像是3通道的，则s.val[0]，s.val[1]，s.val[2]中存储数据
219. \*/
220. Scalar mean;   //均值
221. Scalar stdev;   //标准差
223. //此函数将frame图像best\_box区域的图像片归一化为均值为0的15\*15大小的patch，存在pEx正样本中
224. getPattern(frame(best\_box), pEx, mean, stdev);
226. //Get Fern features on warped patches
227. Mat img;
228. Mat warped;
229. //void GaussianBlur(InputArray src, OutputArray dst, Size ksize, double sigmaX, double sigmaY=0,
230. //                                    int borderType=BORDER\_DEFAULT ) ;
231. //功能：对输入的图像src进行高斯滤波后用dst输出。
232. //src和dst当然分别是输入图像和输出图像。Ksize为高斯滤波器模板大小，sigmaX和sigmaY分别为高斯滤
233. //波在横向和竖向的滤波系数。borderType为边缘扩展点插值类型。
234. //用9\*9高斯核模糊输入帧，存入img  去噪？？
235. GaussianBlur(frame, img, Size(9,9), 1.5);
237. //在img图像中截取bbhull信息（bbhull是包含了位置和大小的矩形框）的图像赋给warped
238. //例如需要提取图像A的某个ROI（感兴趣区域，由矩形框）的话，用Mat类的B=img(ROI)即可提取
239. warped = img(bbhull);
240. RNG& rng = theRNG();  //生成一个随机数
241. Point2f pt(bbhull.x + (bbhull.width-1)\*0.5f, bbhull.y+(bbhull.height-1)\*0.5f);  //取矩形框中心的坐标  int i(2)
243. //nstructs树木（由一个特征组构建，每组特征代表图像块的不同视图表示）的个数
244. //fern[nstructs] nstructs棵树的森林的数组？？
245. vector<**int**> fern(classifier.getNumStructs());
246. pX.clear();
247. Mat patch;
249. //pX为处理后的RectBox最大边界处理后的像素信息，pEx最近邻的RectBox的Pattern，bbP0为最近邻的RectBox。
250. **if** (pX.capacity() < num\_warps \* good\_boxes.size())
251. pX.reserve(num\_warps \* good\_boxes.size());  //pX正样本个数为 仿射变换个数 \* good\_box的个数，故需分配至少这么大的空间
252. **int** idx;
253. **for** (**int** i=0; i< num\_warps; i++){
254. **if** (i>0)
255. //PatchGenerator类用来对图像区域进行仿射变换，先RNG一个随机因子，再调用（）运算符产生一个变换后的正样本。
256. generator(frame, pt, warped, bbhull.size(), rng);
257. **for** (**int** b=0; b < good\_boxes.size(); b++){
258. idx = good\_boxes[b];  //good\_boxes容器保存的是 grid 的索引
259. patch = img(grid[idx]);  //把img的 grid[idx] 区域（也就是bounding box重叠度高的）这一块图像片提取出来
260. //getFeatures函数得到输入的patch的用于树的节点，也就是特征组的特征fern（13位的二进制代码）
261. classifier.getFeatures(patch, grid[idx].sidx, fern);  //grid[idx].sidx 对应的尺度索引
262. pX.push\_back(make\_pair(fern, 1));   //positive ferns <features, labels=1>  正样本
263. }
264. }
265. printf("Positive examples generated: ferns:%d NN:1\n",(**int**)pX.size());
266. }
268. //先对最接近box的RectBox区域得到其patch ,然后将像素信息转换为Pattern，
269. //具体的说就是归一化RectBox对应的patch的size（放缩至patch\_size = 15\*15），将2维的矩阵变成一维的向量信息，
270. //然后将向量信息均值设为0，调整为zero mean and unit variance（ZMUV）
271. //Output: resized Zero-Mean patch
272. **void** TLD::getPattern(**const** Mat& img, Mat& pattern, Scalar& mean, Scalar& stdev){
273. //将img放缩至patch\_size = 15\*15，存到pattern中
274. resize(img, pattern, Size(patch\_size, patch\_size));
276. //计算pattern这个矩阵的均值和标准差
277. //Computes a mean value and a standard deviation of matrix elements.
278. meanStdDev(pattern, mean, stdev);
279. pattern.convertTo(pattern, CV\_32F);
281. //opencv中Mat的运算符有重载， Mat可以 + Mat; + Scalar; + int / float / double 都可以
282. //将矩阵所有元素减去其均值，也就是把patch的均值设为零
283. pattern = pattern - mean.val[0];
284. }
286. /\* Inputs:
287. \* - Image
288. \* - bad\_boxes (Boxes far from the bounding box)
289. \* - variance (pEx variance)
290. \* Outputs
291. \* - Negative fern features (nX)
292. \* - Negative NN examples (nEx)
293. \*/
294. **void** TLD::generateNegativeData(**const** Mat& frame){
295. //由于之前重叠度小于0.2的，都归入 bad\_boxes了，所以数量挺多，下面的函数用于打乱顺序，也就是为了
296. //后面随机选择bad\_boxes
297. random\_shuffle(bad\_boxes.begin(), bad\_boxes.end());//Random shuffle bad\_boxes indexes
298. **int** idx;
299. //Get Fern Features of the boxes with big variance (calculated using integral images)
300. **int** a=0;
301. //int num = std::min((int)bad\_boxes.size(),(int)bad\_patches\*100); //limits the size of bad\_boxes to try
302. printf("negative data generation started.\n");
303. vector<**int**> fern(classifier.getNumStructs());
304. nX.reserve(bad\_boxes.size());
305. Mat patch;
306. **for** (**int** j=0;j<bad\_boxes.size();j++){  //把方差较大的bad\_boxes加入负样本
307. idx = bad\_boxes[j];
308. **if** (getVar(grid[idx],iisum,iisqsum)<var\*0.5f)
309. **continue**;
310. patch =  frame(grid[idx]);
311. classifier.getFeatures(patch, grid[idx].sidx, fern);
312. nX.push\_back(make\_pair(fern, 0)); //得到负样本
313. a++;
314. }
315. printf("Negative examples generated: ferns: %d ", a);
317. //random\_shuffle(bad\_boxes.begin(),bad\_boxes.begin()+bad\_patches);//Randomly selects 'bad\_patches' and get the patterns for NN;
318. Scalar dum1, dum2;
319. //bad\_patches = (int)file["num\_patches"]; 在参数文件中 num\_patches = 100
320. nEx=vector<Mat>(bad\_patches);
321. **for** (**int** i=0;i<bad\_patches;i++){
322. idx=bad\_boxes[i];
323. patch = frame(grid[idx]);
324. //具体的说就是归一化RectBox对应的patch的size（放缩至patch\_size = 15\*15）
325. //由于负样本不需要均值和方差，所以就定义dum，将其舍弃
326. getPattern(patch,nEx[i],dum1,dum2);
327. }
328. printf("NN: %d\n",(**int**)nEx.size());
329. }
331. //该函数通过积分图像计算输入的box的方差
332. **double** TLD::getVar(**const** BoundingBox& box, **const** Mat& sum, **const** Mat& sqsum){
333. **double** brs = sum.at<**int**>(box.y+box.height, box.x+box.width);
334. **double** bls = sum.at<**int**>(box.y+box.height, box.x);
335. **double** trs = sum.at<**int**>(box.y,box.x + box.width);
336. **double** tls = sum.at<**int**>(box.y,box.x);
337. **double** brsq = sqsum.at<**double**>(box.y+box.height,box.x+box.width);
338. **double** blsq = sqsum.at<**double**>(box.y+box.height,box.x);
339. **double** trsq = sqsum.at<**double**>(box.y,box.x+box.width);
340. **double** tlsq = sqsum.at<**double**>(box.y,box.x);
342. **double** mean = (brs+tls-trs-bls)/((**double**)box.area());
343. **double** sqmean = (brsq+tlsq-trsq-blsq)/((**double**)box.area());
344. //方差=E(X^2)-(EX)^2   EX表示均值
345. **return** sqmean-mean\*mean;
346. }
348. **void** TLD::processFrame(**const** cv::Mat& img1,**const** cv::Mat& img2,vector<Point2f>& points1,vector<Point2f>& points2,BoundingBox& bbnext, **bool**& lastboxfound, **bool** tl, **FILE**\* bb\_file){
349. vector<BoundingBox> cbb;
350. vector<**float**> cconf;
351. **int** confident\_detections=0;
352. **int** didx; //detection index
354. ///Track  跟踪模块
355. **if**(lastboxfound && tl){   //tl: train and learn
356. //跟踪
357. track(img1, img2, points1, points2);
358. }
359. **else**{
360. tracked = **false**;
361. }
363. ///Detect   检测模块
364. detect(img2);
366. ///Integration   综合模块
367. //TLD只跟踪单目标，所以综合模块综合跟踪器跟踪到的单个目标和检测器检测到的多个目标，然后只输出保守相似度最大的一个目标
368. **if** (tracked){
369. bbnext=tbb;
370. lastconf=tconf;   //表示相关相似度的阈值
371. lastvalid=tvalid;  //表示保守相似度的阈值
372. printf("Tracked\n");
373. **if**(detected){                                               //   if Detected
374. //通过 重叠度 对检测器检测到的目标bounding box进行聚类，每个类其重叠度小于0.5
375. clusterConf(dbb, dconf, cbb, cconf);                       //   cluster detections
376. printf("Found %d clusters\n",(**int**)cbb.size());
377. **for** (**int** i=0;i<cbb.size();i++){
378. //找到与跟踪器跟踪到的box距离比较远的类（检测器检测到的box），而且它的相关相似度比跟踪器的要大
379. **if** (bbOverlap(tbb, cbb[i])<0.5 && cconf[i]>tconf){  //  Get index of a clusters that is far from tracker and are more confident than the tracker
380. confident\_detections++;  //记录满足上述条件，也就是可信度比较高的目标box的个数
381. didx=i; //detection index
382. }
383. }
384. //如果只有一个满足上述条件的box，那么就用这个目标box来重新初始化跟踪器（也就是用检测器的结果去纠正跟踪器）
385. **if** (confident\_detections==1){                                //if there is ONE such a cluster, re-initialize the tracker
386. printf("Found a better match..reinitializing tracking\n");
387. bbnext=cbb[didx];
388. lastconf=cconf[didx];
389. lastvalid=**false**;
390. }
391. **else** {
392. printf("%d confident cluster was found\n", confident\_detections);
393. **int** cx=0,cy=0,cw=0,ch=0;
394. **int** close\_detections=0;
395. **for** (**int** i=0;i<dbb.size();i++){
396. //找到检测器检测到的box与跟踪器预测到的box距离很近（重叠度大于0.7）的box，对其坐标和大小进行累加
397. **if**(bbOverlap(tbb,dbb[i])>0.7){                     // Get mean of close detections
398. cx += dbb[i].x;
399. cy +=dbb[i].y;
400. cw += dbb[i].width;
401. ch += dbb[i].height;
402. close\_detections++;   //记录最近邻box的个数
403. printf("weighted detection: %d %d %d %d\n",dbb[i].x,dbb[i].y,dbb[i].width,dbb[i].height);
404. }
405. }
406. **if** (close\_detections>0){
407. //对与跟踪器预测到的box距离很近的box 和 跟踪器本身预测到的box 进行坐标与大小的平均作为最终的
408. //目标bounding box，但是跟踪器的权值较大
409. bbnext.x = cvRound((**float**)(10\*tbb.x+cx)/(**float**)(10+close\_detections));   // weighted average trackers trajectory with the close detections
410. bbnext.y = cvRound((**float**)(10\*tbb.y+cy)/(**float**)(10+close\_detections));
411. bbnext.width = cvRound((**float**)(10\*tbb.width+cw)/(**float**)(10+close\_detections));
412. bbnext.height =  cvRound((**float**)(10\*tbb.height+ch)/(**float**)(10+close\_detections));
413. printf("Tracker bb: %d %d %d %d\n",tbb.x,tbb.y,tbb.width,tbb.height);
414. printf("Average bb: %d %d %d %d\n",bbnext.x,bbnext.y,bbnext.width,bbnext.height);
415. printf("Weighting %d close detection(s) with tracker..\n",close\_detections);
416. }
417. **else**{
418. printf("%d close detections were found\n",close\_detections);
420. }
421. }
422. }
423. }
424. **else**{                                       //   If NOT tracking
425. printf("Not tracking..\n");
426. lastboxfound = **false**;
427. lastvalid = **false**;
428. //如果跟踪器没有跟踪到目标，但是检测器检测到了一些可能的目标box，那么同样对其进行聚类，但只是简单的
429. //将聚类的cbb[0]作为新的跟踪目标box（不比较相似度了？？还是里面已经排好序了？？），重新初始化跟踪器
430. **if**(detected){                           //  and detector is defined
431. clusterConf(dbb,dconf,cbb,cconf);   //  cluster detections
432. printf("Found %d clusters\n",(**int**)cbb.size());
433. **if** (cconf.size()==1){
434. bbnext=cbb[0];
435. lastconf=cconf[0];
436. printf("Confident detection..reinitializing tracker\n");
437. lastboxfound = **true**;
438. }
439. }
440. }
441. lastbox=bbnext;
442. **if** (lastboxfound)
443. fprintf(bb\_file,"%d,%d,%d,%d,%f\n",lastbox.x,lastbox.y,lastbox.br().x,lastbox.br().y,lastconf);
444. **else**
445. fprintf(bb\_file,"NaN,NaN,NaN,NaN,NaN\n");
447. ///learn 学习模块
448. **if** (lastvalid && tl)
449. learn(img2);
450. }
452. /\*Inputs:
453. \* -current frame(img2), last frame(img1), last Bbox(bbox\_f[0]).
454. \*Outputs:
455. \*- Confidence(tconf), Predicted bounding box(tbb), Validity(tvalid), points2 (for display purposes only)
456. \*/
457. **void** TLD::track(**const** Mat& img1, **const** Mat& img2, vector<Point2f>& points1, vector<Point2f>& points2){
459. //Generate points
460. //网格均匀撒点（均匀采样），在lastbox中共产生最多10\*10=100个特征点，存于points1
461. bbPoints(points1, lastbox);
462. **if** (points1.size()<1){
463. printf("BB= %d %d %d %d, Points not generated\n",lastbox.x,lastbox.y,lastbox.width,lastbox.height);
464. tvalid=**false**;
465. tracked=**false**;
466. **return**;
467. }
468. vector<Point2f> points = points1;
470. //Frame-to-frame tracking with forward-backward error cheking
471. //trackf2f函数完成：跟踪、计算FB error和匹配相似度sim，然后筛选出 FB\_error[i] <= median(FB\_error) 和
472. //sim\_error[i] > median(sim\_error) 的特征点（跟踪结果不好的特征点），剩下的是不到50%的特征点
473. tracked = tracker.trackf2f(img1, img2, points, points2);
474. **if** (tracked){
475. //Bounding box prediction
476. //利用剩下的这不到一半的跟踪点输入来预测bounding box在当前帧的位置和大小 tbb
477. bbPredict(points, points2, lastbox, tbb);
478. //跟踪失败检测：如果FB error的中值大于10个像素（经验值），或者预测到的当前box的位置移出图像，则
479. //认为跟踪错误，此时不返回bounding box；Rect::br()返回的是右下角的坐标
480. //getFB()返回的是FB error的中值
481. **if** (tracker.getFB()>10 || tbb.x>img2.cols ||  tbb.y>img2.rows || tbb.br().x < 1 || tbb.br().y <1){
482. tvalid =**false**; //too unstable prediction or bounding box out of image
483. tracked = **false**;
484. printf("Too unstable predictions FB error=%f\n", tracker.getFB());
485. **return**;
486. }
488. //Estimate Confidence and Validity
489. //评估跟踪确信度和有效性
490. Mat pattern;
491. Scalar mean, stdev;
492. BoundingBox bb;
493. bb.x = max(tbb.x,0);
494. bb.y = max(tbb.y,0);
495. bb.width = min(min(img2.cols-tbb.x,tbb.width), min(tbb.width, tbb.br().x));
496. bb.height = min(min(img2.rows-tbb.y,tbb.height),min(tbb.height,tbb.br().y));
497. //归一化img2(bb)对应的patch的size（放缩至patch\_size = 15\*15），存入pattern
498. getPattern(img2(bb),pattern,mean,stdev);
499. vector<**int**> isin;
500. **float** dummy;
501. //计算图像片pattern到在线模型M的保守相似度
502. classifier.NNConf(pattern,isin,dummy,tconf); //Conservative Similarity
503. tvalid = lastvalid;
504. //保守相似度大于阈值，则评估跟踪有效
505. **if** (tconf>classifier.thr\_nn\_valid){
506. tvalid =**true**;
507. }
508. }
509. **else**
510. printf("No points tracked\n");
512. }
514. //网格均匀撒点，box共10\*10=100个特征点
515. **void** TLD::bbPoints(vector<cv::Point2f>& points, **const** BoundingBox& bb){
516. **int** max\_pts=10;
517. **int** margin\_h=0; //采样边界
518. **int** margin\_v=0;
519. //网格均匀撒点
520. **int** stepx = ceil((bb.width-2\*margin\_h)/max\_pts);  //ceil返回大于或者等于指定表达式的最小整数
521. **int** stepy = ceil((bb.height-2\*margin\_v)/max\_pts);
522. //网格均匀撒点，box共10\*10=100个特征点
523. **for** (**int** y=bb.y+margin\_v; y<bb.y+bb.height-margin\_v; y+=stepy){
524. **for** (**int** x=bb.x+margin\_h;x<bb.x+bb.width-margin\_h;x+=stepx){
525. points.push\_back(Point2f(x,y));
526. }
527. }
528. }
530. //利用剩下的这不到一半的跟踪点输入来预测bounding box在当前帧的位置和大小
531. **void** TLD::bbPredict(**const** vector<cv::Point2f>& points1,**const** vector<cv::Point2f>& points2,
532. **const** BoundingBox& bb1,BoundingBox& bb2)    {
533. **int** npoints = (**int**)points1.size();
534. vector<**float**> xoff(npoints);  //位移
535. vector<**float**> yoff(npoints);
536. printf("tracked points : %d\n", npoints);
537. **for** (**int** i=0;i<npoints;i++){   //计算每个特征点在两帧之间的位移
538. xoff[i]=points2[i].x - points1[i].x;
539. yoff[i]=points2[i].y - points1[i].y;
540. }
541. **float** dx = median(xoff);   //计算位移的中值
542. **float** dy = median(yoff);
543. **float** s;
544. //计算bounding box尺度scale的变化：通过计算 当前特征点相互间的距离 与 先前（上一帧）特征点相互间的距离 的
545. //比值，以比值的中值作为尺度的变化因子
546. **if** (npoints>1){
547. vector<**float**> d;
548. d.reserve(npoints\*(npoints-1)/2);  //等差数列求和：1+2+...+(npoints-1)
549. **for** (**int** i=0;i<npoints;i++){
550. **for** (**int** j=i+1;j<npoints;j++){
551. //计算 当前特征点相互间的距离 与 先前（上一帧）特征点相互间的距离 的比值（位移用绝对值）
552. d.push\_back(norm(points2[i]-points2[j])/norm(points1[i]-points1[j]));
553. }
554. }
555. s = median(d);
556. }
557. **else** {
558. s = 1.0;
559. }
561. **float** s1 = 0.5\*(s-1)\*bb1.width;
562. **float** s2 = 0.5\*(s-1)\*bb1.height;
563. printf("s= %f s1= %f s2= %f \n", s, s1, s2);
565. //得到当前bounding box的位置与大小信息
566. //当前box的x坐标 = 前一帧box的x坐标 + 全部特征点位移的中值（可理解为box移动近似的位移） - 当前box宽的一半
567. bb2.x = round( bb1.x + dx - s1);
568. bb2.y = round( bb1.y + dy -s2);
569. bb2.width = round(bb1.width\*s);
570. bb2.height = round(bb1.height\*s);
571. printf("predicted bb: %d %d %d %d\n",bb2.x,bb2.y,bb2.br().x,bb2.br().y);
572. }
574. **void** TLD::detect(**const** cv::Mat& frame){
575. //cleaning
576. dbb.clear();
577. dconf.clear();
578. dt.bb.clear();
579. //GetTickCount返回从操作系统启动到现在所经过的时间
580. **double** t = (**double**)getTickCount();
581. Mat img(frame.rows, frame.cols, CV\_8U);
582. integral(frame,iisum,iisqsum);   //计算frame的积分图
583. GaussianBlur(frame,img,Size(9,9),1.5);  //高斯模糊，去噪？
584. **int** numtrees = classifier.getNumStructs();
585. **float** fern\_th = classifier.getFernTh(); //getFernTh()返回thr\_fern; 集合分类器的分类阈值
586. vector <**int**> ferns(10);
587. **float** conf;
588. **int** a=0;
589. Mat patch;
590. //级联分类器模块一：方差检测模块，利用积分图计算每个待检测窗口的方差，方差大于var阈值（目标patch方差的50%）的，
591. //则认为其含有前景目标
592. **for** (**int** i=0; i<grid.size(); i++){  //FIXME: BottleNeck 瓶颈
593. **if** (getVar(grid[i],iisum,iisqsum) >= var){  //计算每一个扫描窗口的方差
594. a++;
595. //级联分类器模块二：集合分类器检测模块
596. patch = img(grid[i]);
597. classifier.getFeatures(patch,grid[i].sidx,ferns); //得到该patch特征（13位的二进制代码）
598. conf = classifier.measure\_forest(ferns);  //计算该特征值对应的后验概率累加值
599. tmp.conf[i]=conf;   //Detector data中定义TempStruct tmp;
600. tmp.patt[i]=ferns;
601. //如果集合分类器的后验概率的平均值大于阈值fern\_th（由训练得到），就认为含有前景目标
602. **if** (conf > numtrees\*fern\_th){
603. dt.bb.push\_back(i);  //将通过以上两个检测模块的扫描窗口记录在detect structure中
604. }
605. }
606. **else**
607. tmp.conf[i]=0.0;
608. }
609. **int** detections = dt.bb.size();
610. printf("%d Bounding boxes passed the variance filter\n",a);
611. printf("%d Initial detection from Fern Classifier\n", detections);
613. //如果通过以上两个检测模块的扫描窗口数大于100个，则只取后验概率大的前100个
614. **if** (detections>100){   //CComparator(tmp.conf)指定比较方式？？？
615. nth\_element(dt.bb.begin(), dt.bb.begin()+100, dt.bb.end(), CComparator(tmp.conf));
616. dt.bb.resize(100);
617. detections=100;
618. }
619. //  for (int i=0;i<detections;i++){
620. //        drawBox(img,grid[dt.bb[i]]);
621. //    }
622. //  imshow("detections",img);
623. **if** (detections==0){
624. detected=**false**;
625. **return**;
626. }
627. printf("Fern detector made %d detections ",detections);
629. //两次使用getTickCount()，然后再除以getTickFrequency()，计算出来的是以秒s为单位的时间（opencv 2.0 以前是ms）
630. t=(**double**)getTickCount()-t;
631. printf("in %gms\n", t\*1000/getTickFrequency());  //打印以上代码运行使用的毫秒数
633. //  Initialize detection structure
634. dt.patt = vector<vector<**int**> >(detections,vector<**int**>(10,0));        //  Corresponding codes of the Ensemble Classifier
635. dt.conf1 = vector<**float**>(detections);                                //  Relative Similarity (for final nearest neighbour classifier)
636. dt.conf2 =vector<**float**>(detections);                                 //  Conservative Similarity (for integration with tracker)
637. dt.isin = vector<vector<**int**> >(detections,vector<**int**>(3,-1));        //  Detected (isin=1) or rejected (isin=0) by nearest neighbour classifier
638. dt.patch = vector<Mat>(detections,Mat(patch\_size,patch\_size,CV\_32F));//  Corresponding patches
639. **int** idx;
640. Scalar mean, stdev;
641. **float** nn\_th = classifier.getNNTh();
642. //级联分类器模块三：最近邻分类器检测模块
643. **for** (**int** i=0;i<detections;i++){                                         //  for every remaining detection
644. idx=dt.bb[i];                                                       //  Get the detected bounding box index
645. patch = frame(grid[idx]);
646. getPattern(patch,dt.patch[i],mean,stdev);                //  Get pattern within bounding box
647. //计算图像片pattern到在线模型M的相关相似度和保守相似度
648. classifier.NNConf(dt.patch[i],dt.isin[i],dt.conf1[i],dt.conf2[i]);  //  Evaluate nearest neighbour classifier
649. dt.patt[i]=tmp.patt[idx];
650. //printf("Testing feature %d, conf:%f isin:(%d|%d|%d)\n",i,dt.conf1[i],dt.isin[i][0],dt.isin[i][1],dt.isin[i][2]);
651. //相关相似度大于阈值，则认为含有前景目标
652. **if** (dt.conf1[i]>nn\_th){                                               //  idx = dt.conf1 > tld.model.thr\_nn; % get all indexes that made it through the nearest neighbour
653. dbb.push\_back(grid[idx]);                                         //  BB    = dt.bb(:,idx); % bounding boxes
654. dconf.push\_back(dt.conf2[i]);                                     //  Conf  = dt.conf2(:,idx); % conservative confidences
655. }
656. }
657. //打印检测到的可能存在目标的扫描窗口数（可以通过三个级联检测器的）
658. **if** (dbb.size()>0){
659. printf("Found %d NN matches\n",(**int**)dbb.size());
660. detected=**true**;
661. }
662. **else**{
663. printf("No NN matches found.\n");
664. detected=**false**;
665. }
666. }
668. //作者已经用python脚本../datasets/evaluate\_vis.py来完成算法评估功能，具体见README
669. **void** TLD::evaluate(){
670. }
672. **void** TLD::learn(**const** Mat& img){
673. printf("[Learning] ");
675. ///Check consistency
676. //检测一致性
677. BoundingBox bb;
678. bb.x = max(lastbox.x,0);
679. bb.y = max(lastbox.y,0);
680. bb.width = min(min(img.cols-lastbox.x,lastbox.width),min(lastbox.width,lastbox.br().x));
681. bb.height = min(min(img.rows-lastbox.y,lastbox.height),min(lastbox.height,lastbox.br().y));
682. Scalar mean, stdev;
683. Mat pattern;
684. //归一化img(bb)对应的patch的size（放缩至patch\_size = 15\*15），存入pattern
685. getPattern(img(bb), pattern, mean, stdev);
686. vector<**int**> isin;
687. **float** dummy, conf;
688. //计算输入图像片（跟踪器的目标box）与在线模型之间的相关相似度conf
689. classifier.NNConf(pattern,isin,conf,dummy);
690. **if** (conf<0.5) {   //如果相似度太小了，就不训练
691. printf("Fast change..not training\n");
692. lastvalid =**false**;
693. **return**;
694. }
695. **if** (pow(stdev.val[0], 2)< var){  //如果方差太小了，也不训练
696. printf("Low variance..not training\n");
697. lastvalid=**false**;
698. **return**;
699. }
700. **if**(isin[2]==1){   //如果被被识别为负样本，也不训练
701. printf("Patch in negative data..not traing");
702. lastvalid=**false**;
703. **return**;
704. }
706. /// Data generation  样本产生
707. **for** (**int** i=0;i<grid.size();i++){   //计算所有的扫描窗口与目标box的重叠度
708. grid[i].overlap = bbOverlap(lastbox, grid[i]);
709. }
710. //集合分类器
711. vector<pair<vector<**int**>,**int**> > fern\_examples;
712. good\_boxes.clear();
713. bad\_boxes.clear();
714. //此函数根据传入的lastbox，在整帧图像中的全部窗口中寻找与该lastbox距离最小（即最相似，
715. //重叠度最大）的num\_closest\_update个窗口，然后把这些窗口 归入good\_boxes容器（只是把网格数组的索引存入）
716. //同时，把重叠度小于0.2的，归入 bad\_boxes 容器
717. getOverlappingBoxes(lastbox, num\_closest\_update);
718. **if** (good\_boxes.size()>0)
719. generatePositiveData(img, num\_warps\_update);  //用仿射模型产生正样本（类似于第一帧的方法，但只产生10\*10=100个）
720. **else**{
721. lastvalid = **false**;
722. printf("No good boxes..Not training");
723. **return**;
724. }
725. fern\_examples.reserve(pX.size() + bad\_boxes.size());
726. fern\_examples.assign(pX.begin(), pX.end());
727. **int** idx;
728. **for** (**int** i=0;i<bad\_boxes.size();i++){
729. idx=bad\_boxes[i];
730. **if** (tmp.conf[idx]>=1){   //加入负样本，相似度大于1？？相似度不是出于0和1之间吗？
731. fern\_examples.push\_back(make\_pair(tmp.patt[idx],0));
732. }
733. }
734. //最近邻分类器
735. vector<Mat> nn\_examples;
736. nn\_examples.reserve(dt.bb.size()+1);
737. nn\_examples.push\_back(pEx);
738. **for** (**int** i=0;i<dt.bb.size();i++){
739. idx = dt.bb[i];
740. **if** (bbOverlap(lastbox,grid[idx]) < bad\_overlap)
741. nn\_examples.push\_back(dt.patch[i]);
742. }
744. /// Classifiers update  分类器训练
745. classifier.trainF(fern\_examples,2);
746. classifier.trainNN(nn\_examples);
747. classifier.show(); //把正样本库（在线模型）包含的所有正样本显示在窗口上
748. }
750. //检测器采用扫描窗口的策略
751. //此函数根据传入的box（目标边界框）在传入的图像中构建全部的扫描窗口，并计算每个窗口与box的重叠度
752. **void** TLD::buildGrid(**const** cv::Mat& img, **const** cv::Rect& box){
753. **const** **float** SHIFT = 0.1;  //扫描窗口步长为 宽高的 10%
754. //尺度缩放系数为1.2 （0.16151\*1.2=0.19381），共21种尺度变换
755. **const** **float** SCALES[] = {0.16151,0.19381,0.23257,0.27908,0.33490,0.40188,0.48225,
756. 0.57870,0.69444,0.83333,1,1.20000,1.44000,1.72800,
757. 2.07360,2.48832,2.98598,3.58318,4.29982,5.15978,6.19174};
758. **int** width, height, min\_bb\_side;
759. //Rect bbox;
760. BoundingBox bbox;
761. Size scale;
762. **int** sc=0;
764. **for** (**int** s=0; s < 21; s++){
765. width = round(box.width\*SCALES[s]);
766. height = round(box.height\*SCALES[s]);
767. min\_bb\_side = min(height,width);  //bounding box最短的边
768. //由于图像片（min\_win 为15x15像素）是在bounding box中采样得到的，所以box必须比min\_win要大
769. //另外，输入的图像肯定得比 bounding box 要大了
770. **if** (min\_bb\_side < min\_win || width > img.cols || height > img.rows)
771. **continue**;
772. scale.width = width;
773. scale.height = height;
774. //push\_back在vector类中作用为在vector尾部加入一个数据
775. //scales在类TLD中定义：std::vector<cv::Size> scales;
776. scales.push\_back(scale);  //把该尺度的窗口存入scales容器，避免在扫描时计算，加快检测速度
777. **for** (**int** y=1; y<img.rows-height; y+=round(SHIFT\*min\_bb\_side)){  //按步长移动窗口
778. **for** (**int** x=1; x<img.cols-width; x+=round(SHIFT\*min\_bb\_side)){
779. bbox.x = x;
780. bbox.y = y;
781. bbox.width = width;
782. bbox.height = height;
783. //判断传入的bounding box（目标边界框）与 传入图像中的此时窗口的 重叠度，
784. //以此来确定该图像窗口是否含有目标
785. bbox.overlap = bbOverlap(bbox, BoundingBox(box));
786. bbox.sidx = sc;  //属于第几个尺度
787. //grid在类TLD中定义：std::vector<BoundingBox> grid;
788. //把本位置和本尺度的扫描窗口存入grid容器
789. grid.push\_back(bbox);
790. }
791. }
792. sc++;
793. }
794. }
796. //此函数计算两个bounding box 的重叠度
797. //重叠度定义为 两个box的交集 与 它们的并集 的比
798. **float** TLD::bbOverlap(**const** BoundingBox& box1, **const** BoundingBox& box2){
799. //先判断坐标，假如它们都没有重叠的地方，就直接返回0
800. **if** (box1.x > box2.x + box2.width) { **return** 0.0; }
801. **if** (box1.y > box2.y + box2.height) { **return** 0.0; }
802. **if** (box1.x + box1.width < box2.x) { **return** 0.0; }
803. **if** (box1.y + box1.height < box2.y) { **return** 0.0; }
805. **float** colInt =  min(box1.x + box1.width, box2.x + box2.width) - max(box1.x, box2.x);
806. **float** rowInt =  min(box1.y + box1.height, box2.y + box2.height) - max(box1.y, box2.y);
808. **float** intersection = colInt \* rowInt;
809. **float** area1 = box1.width \* box1.height;
810. **float** area2 = box2.width \* box2.height;
811. **return** intersection / (area1 + area2 - intersection);
812. }
814. //此函数根据传入的box1（目标边界框），在整帧图像中的全部窗口中寻找与该box1距离最小（即最相似，
815. //重叠度最大）的num\_closest个窗口，然后把这些窗口 归入good\_boxes容器（只是把网格数组的索引存入）
816. //同时，把重叠度小于0.2的，归入 bad\_boxes 容器
817. **void** TLD::getOverlappingBoxes(**const** cv::Rect& box1,**int** num\_closest){
818. **float** max\_overlap = 0;
819. **for** (**int** i=0;i<grid.size();i++){
820. **if** (grid[i].overlap > max\_overlap) {  //找出重叠度最大的box
821. max\_overlap = grid[i].overlap;
822. best\_box = grid[i];
823. }
824. **if** (grid[i].overlap > 0.6){   //重叠度大于0.6的，归入 good\_boxes
825. good\_boxes.push\_back(i);
826. }
827. **else** **if** (grid[i].overlap < bad\_overlap){  //重叠度小于0.2的，归入 bad\_boxes
828. bad\_boxes.push\_back(i);
829. }
830. }
831. //Get the best num\_closest (10) boxes and puts them in good\_boxes
832. **if** (good\_boxes.size()>num\_closest){
833. //STL中的nth\_element()方法找出一个数列中排名第n（下面为第num\_closest）的那个数。这个函数运行后
834. //在good\_boxes[num\_closest]前面num\_closest个数都比他大，也就是找到最好的num\_closest个box了
835. std::nth\_element(good\_boxes.begin(), good\_boxes.begin() + num\_closest, good\_boxes.end(), OComparator(grid));
836. //重新压缩good\_boxes为num\_closest大小
837. good\_boxes.resize(num\_closest);
838. }
839. //获取good\_boxes 的 Hull壳，也就是窗口的边框
840. getBBHull();
841. }
843. //此函数获取good\_boxes 的 Hull壳，也就是窗口（图像）的边框 bounding box
844. **void** TLD::getBBHull(){
845. **int** x1=INT\_MAX, x2=0;  //INT\_MAX 最大的整形数
846. **int** y1=INT\_MAX, y2=0;
847. **int** idx;
848. **for** (**int** i=0;i<good\_boxes.size();i++){
849. idx= good\_boxes[i];
850. x1=min(grid[idx].x,x1);   //防止出现负数？？
851. y1=min(grid[idx].y,y1);
852. x2=max(grid[idx].x + grid[idx].width,x2);
853. y2=max(grid[idx].y + grid[idx].height,y2);
854. }
855. bbhull.x = x1;
856. bbhull.y = y1;
857. bbhull.width = x2-x1;
858. bbhull.height = y2 -y1;
859. }
861. //如果两个box的重叠度小于0.5，返回false，否则返回true
862. **bool** bbcomp(**const** BoundingBox& b1,**const** BoundingBox& b2){
863. TLD t;
864. **if** (t.bbOverlap(b1,b2)<0.5)
865. **return** **false**;
866. **else**
867. **return** **true**;
868. }
870. **int** TLD::clusterBB(**const** vector<BoundingBox>& dbb,vector<**int**>& indexes){
871. //FIXME: Conditional jump or move depends on uninitialised value(s)
872. **const** **int** c = dbb.size();
873. //1. Build proximity matrix
874. Mat D(c,c,CV\_32F);
875. **float** d;
876. **for** (**int** i=0;i<c;i++){
877. **for** (**int** j=i+1;j<c;j++){
878. d = 1-bbOverlap(dbb[i],dbb[j]);
879. D.at<**float**>(i,j) = d;
880. D.at<**float**>(j,i) = d;
881. }
882. }
883. //2. Initialize disjoint clustering
884. **float** L[c-1]; //Level
885. **int** nodes[c-1][2];
886. **int** belongs[c];
887. **int** m=c;
888. **for** (**int** i=0;i<c;i++){
889. belongs[i]=i;
890. }
891. **for** (**int** it=0;it<c-1;it++){
892. //3. Find nearest neighbor
893. **float** min\_d = 1;
894. **int** node\_a, node\_b;
895. **for** (**int** i=0;i<D.rows;i++){
896. **for** (**int** j=i+1;j<D.cols;j++){
897. **if** (D.at<**float**>(i,j)<min\_d && belongs[i]!=belongs[j]){
898. min\_d = D.at<**float**>(i,j);
899. node\_a = i;
900. node\_b = j;
901. }
902. }
903. }
904. **if** (min\_d>0.5){
905. **int** max\_idx =0;
906. **bool** visited;
907. **for** (**int** j=0;j<c;j++){
908. visited = **false**;
909. **for**(**int** i=0;i<2\*c-1;i++){
910. **if** (belongs[j]==i){
911. indexes[j]=max\_idx;
912. visited = **true**;
913. }
914. }
915. **if** (visited)
916. max\_idx++;
917. }
918. **return** max\_idx;
919. }
921. //4. Merge clusters and assign level
922. L[m]=min\_d;
923. nodes[it][0] = belongs[node\_a];
924. nodes[it][1] = belongs[node\_b];
925. **for** (**int** k=0;k<c;k++){
926. **if** (belongs[k]==belongs[node\_a] || belongs[k]==belongs[node\_b])
927. belongs[k]=m;
928. }
929. m++;
930. }
931. **return** 1;
933. }
935. //对检测器检测到的目标bounding box进行聚类
936. //聚类（Cluster）分析是由若干模式（Pattern）组成的，通常，模式是一个度量（Measurement）的向量，或者是多维空间中的
937. //一个点。聚类分析以相似性为基础，在一个聚类中的模式之间比不在同一聚类中的模式之间具有更多的相似性。
938. **void** TLD::clusterConf(**const** vector<BoundingBox>& dbb,**const** vector<**float**>& dconf,vector<BoundingBox>& cbb,vector<**float**>& cconf){
939. **int** numbb =dbb.size();
940. vector<**int**> T;
941. **float** space\_thr = 0.5;
942. **int** c=1;    //记录 聚类的类个数
943. **switch** (numbb){  //检测到的含有目标的bounding box个数
944. **case** 1:
945. cbb=vector<BoundingBox>(1,dbb[0]);  //如果只检测到一个，那么这个就是检测器检测到的目标
946. cconf=vector<**float**>(1,dconf[0]);
947. **return**;
948. **break**;
949. **case** 2:
950. T =vector<**int**>(2,0);
951. //此函数计算两个bounding box 的重叠度
952. **if** (1 - bbOverlap(dbb[0],dbb[1]) > space\_thr){  //如果只检测到两个box，但他们的重叠度小于0.5
953. T[1]=1;
954. c=2;  //重叠度小于0.5的box，属于不同的类
955. }
956. **break**;
957. **default**:  //检测到的box数目大于2个，则筛选出重叠度大于0.5的
958. T = vector<**int**>(numbb, 0);
959. //stable\_partition()重新排列元素，使得满足指定条件的元素排在不满足条件的元素前面。它维持着两组元素的顺序关系。
960. //STL partition就是把一个区间中的元素按照某个条件分成两类。返回第二类子集的起点
961. //bbcomp()函数判断两个box的重叠度小于0.5，返回false，否则返回true （分界点是重叠度：0.5）
962. //partition() 将dbb划分为两个子集，将满足两个box的重叠度小于0.5的元素移动到序列的前面，为一个子集，重叠度大于0.5的，
963. //放在序列后面，为第二个子集，但两个子集的大小不知道，返回第二类子集的起点
964. c = partition(dbb, T, (\*bbcomp));   //重叠度小于0.5的box，属于不同的类，所以c是不同的类别个数
965. //c = clusterBB(dbb,T);
966. **break**;
967. }
969. cconf=vector<**float**>(c);
970. cbb=vector<BoundingBox>(c);
971. printf("Cluster indexes: ");
972. BoundingBox bx;
973. **for** (**int** i=0;i<c;i++){   //类别个数
974. **float** cnf=0;
975. **int** N=0,mx=0,my=0,mw=0,mh=0;
976. **for** (**int** j=0;j<T.size();j++){  //检测到的bounding box个数
977. **if** (T[j]==i){   //将聚类为同一个类别的box的坐标和大小进行累加
978. printf("%d ",i);
979. cnf=cnf+dconf[j];
980. mx=mx+dbb[j].x;
981. my=my+dbb[j].y;
982. mw=mw+dbb[j].width;
983. mh=mh+dbb[j].height;
984. N++;
985. }
986. }
987. **if** (N>0){   //然后求该类的box的坐标和大小的平均值，将平均值作为该类的box的代表
988. cconf[i]=cnf/N;
989. bx.x=cvRound(mx/N);
990. bx.y=cvRound(my/N);
991. bx.width=cvRound(mw/N);
992. bx.height=cvRound(mh/N);
993. cbb[i]=bx;  //返回的是聚类，每一个类都有一个代表的bounding box
994. }
995. }
996. printf("\n");
997. }

## FerNNClassifier

下面是自己在看论文和这些大牛的分析过程中，对代码进行了一些理解，但是由于自己接触图像处理和机器视觉没多久，另外由于自己编程能力比较弱，所以分析过程可能会有不少的错误，希望各位不吝指正。而且，因为编程很多地方不懂，所以注释得非常乱，还海涵。

FerNNClassifier.h

**[cpp]** [view plaincopy](http://blog.csdn.net/zouxy09/article/details/7893090)

1. /\*
2. \* FerNNClassifier.h
3. \*
4. \*  Created on: Jun 14, 2011
5. \*      Author: alantrrs
6. \*/
8. #include <opencv2/opencv.hpp>
9. #include <stdio.h>
10. **class** FerNNClassifier{
11. **private**:
12. //下面这些参数通过程序开始运行时读入parameters.yml文件进行初始化
13. **float** thr\_fern;
14. **int** structSize;
15. **int** nstructs;
16. **float** valid;
17. **float** ncc\_thesame;
18. **float** thr\_nn;
19. **int** acum;
20. **public**:
21. //Parameters
22. **float** thr\_nn\_valid;
24. **void** read(**const** cv::FileNode& file);
25. **void** prepare(**const** std::vector<cv::Size>& scales);
26. **void** getFeatures(**const** cv::Mat& image,**const** **int**& scale\_idx,std::vector<**int**>& fern);
27. **void** update(**const** std::vector<**int**>& fern, **int** C, **int** N);
28. **float** measure\_forest(std::vector<**int**> fern);
29. **void** trainF(**const** std::vector<std::pair<std::vector<**int**>,**int**> >& ferns,**int** resample);
30. **void** trainNN(**const** std::vector<cv::Mat>& nn\_examples);
31. **void** NNConf(**const** cv::Mat& example,std::vector<**int**>& isin,**float**& rsconf,**float**& csconf);
32. **void** evaluateTh(**const** std::vector<std::pair<std::vector<**int**>,**int**> >& nXT,**const** std::vector<cv::Mat>& nExT);
33. **void** show();
34. //Ferns Members
35. **int** getNumStructs(){**return** nstructs;}
36. **float** getFernTh(){**return** thr\_fern;}
37. **float** getNNTh(){**return** thr\_nn;}
39. **struct** Feature   //特征结构体
40. {
41. uchar x1, y1, x2, y2;
42. Feature() : x1(0), y1(0), x2(0), y2(0) {}
43. Feature(**int** \_x1, **int** \_y1, **int** \_x2, **int** \_y2)
44. : x1((uchar)\_x1), y1((uchar)\_y1), x2((uchar)\_x2), y2((uchar)\_y2)
45. {}
46. **bool** operator ()(**const** cv::Mat& patch) **const**
47. {
48. //二维单通道元素可以用Mat::at(i, j)访问，i是行序号，j是列序号
49. //返回的patch图像片在(y1,x1)和(y2, x2)点的像素比较值，返回0或者1
50. **return** patch.at<uchar>(y1,x1) > patch.at<uchar>(y2, x2);
51. }
52. };
53. //Ferns（蕨类植物：有根、茎、叶之分，不具花）features 特征组？
54. std::vector<std::vector<Feature> > features; //Ferns features (one std::vector for each scale)
55. std::vector< std::vector<**int**> > nCounter; //negative counter
56. std::vector< std::vector<**int**> > pCounter; //positive counter
57. std::vector< std::vector<**float**> > posteriors; //Ferns posteriors
58. **float** thrN; //Negative threshold
59. **float** thrP;  //Positive thershold
61. //NN Members
62. std::vector<cv::Mat> pEx; //NN positive examples
63. std::vector<cv::Mat> nEx; //NN negative examples
64. };

FerNNClassifier.cpp

**[cpp]** [view plaincopy](http://blog.csdn.net/zouxy09/article/details/7893090)

1. /\*
2. \* FerNNClassifier.cpp
3. \*
4. \*  Created on: Jun 14, 2011
5. \*      Author: alantrrs
6. \*/
8. #include <FerNNClassifier.h>
10. **using** **namespace** cv;
11. **using** **namespace** std;
13. **void** FerNNClassifier::read(**const** FileNode& file){
14. ///Classifier Parameters
15. //下面这些参数通过程序开始运行时读入parameters.yml文件进行初始化
16. valid = (**float**)file["valid"];
17. ncc\_thesame = (**float**)file["ncc\_thesame"];
18. nstructs = (**int**)file["num\_trees"];   //树木（由一个特征组构建，每组特征代表图像块的不同视图表示）的个数
19. structSize = (**int**)file["num\_features"];  //每棵树的特征个数，也即每棵树的节点个数；树上每一个特征都作为一个决策节点
20. thr\_fern = (**float**)file["thr\_fern"];
21. thr\_nn = (**float**)file["thr\_nn"];
22. thr\_nn\_valid = (**float**)file["thr\_nn\_valid"];
23. }
25. **void** FerNNClassifier::prepare(**const** vector<Size>& scales){
26. acum = 0;
27. //Initialize test locations for features
28. **int** totalFeatures = nstructs \* structSize;
29. //二维向量  包含全部尺度（scales）的扫描窗口，每个尺度包含totalFeatures个特征
30. features = vector<vector<Feature> >(scales.size(), vector<Feature> (totalFeatures));
32. //opencv中自带的一个随机数发生器的类RNG
33. RNG& rng = theRNG();
35. **float** x1f,x2f,y1f,y2f;
36. **int** x1, x2, y1, y2;
37. //集合分类器基于n个基本分类器，每个分类器都是基于一个pixel comparisons（像素比较集）的；
38. //pixel comparisons的产生方法：先用一个归一化的patch去离散化像素空间，产生所有可能的垂直和水平的pixel comparisons
39. //然后我们把这些pixel comparisons随机分配给n个分类器，每个分类器得到完全不同的pixel comparisons（特征集合），
40. //这样，所有分类器的特征组统一起来就可以覆盖整个patch了
42. //用随机数去填充每一个尺度扫描窗口的特征
43. **for** (**int** i=0;i<totalFeatures;i++){
44. x1f = (**float**)rng;
45. y1f = (**float**)rng;
46. x2f = (**float**)rng;
47. y2f = (**float**)rng;
48. **for** (**int** s=0; s<scales.size(); s++){
49. x1 = x1f \* scales[s].width;
50. y1 = y1f \* scales[s].height;
51. x2 = x2f \* scales[s].width;
52. y2 = y2f \* scales[s].height;
53. //第s种尺度的第i个特征  两个随机分配的像素点坐标
54. features[s][i] = Feature(x1, y1, x2, y2);
55. }
56. }
57. //Thresholds
58. thrN = 0.5 \* nstructs;
60. //Initialize Posteriors  初始化后验概率
61. //后验概率指每一个分类器对传入的图像片进行像素对比，每一个像素对比得到0或者1，所有的特征13个comparison对比，
62. //连成一个13位的二进制代码x，然后索引到一个记录了后验概率的数组P(y|x)，y为0或者1（二分类），也就是出现x的
63. //基础上，该图像片为y的概率是多少对n个基本分类器的后验概率做平均，大于0.5则判定其含有目标
64. **for** (**int** i = 0; i<nstructs; i++) {
65. //每一个每类器维护一个后验概率的分布，这个分布有2^d个条目（entries），这里d是像素比较pixel comparisons
66. //的个数，这里是structSize，即13个comparison，所以会产生2^13即8,192个可能的code，每一个code对应一个后验概率
67. //后验概率P(y|x)= #p/(#p+#n) ,#p和#n分别是正和负图像片的数目，也就是下面的pCounter和nCounter
68. //初始化时，每个后验概率都得初始化为0；运行时候以下面方式更新：已知类别标签的样本（训练样本）通过n个分类器
69. //进行分类，如果分类结果错误，那么响应的#p和#n就会更新，这样P(y|x)也相应更新了
70. posteriors.push\_back(vector<**float**>(pow(2.0,structSize), 0));
71. pCounter.push\_back(vector<**int**>(pow(2.0,structSize), 0));
72. nCounter.push\_back(vector<**int**>(pow(2.0,structSize), 0));
73. }
74. }
76. //该函数得到输入的image的用于树的节点，也就是特征组的特征（13位的二进制代码）
77. **void** FerNNClassifier::getFeatures(**const** cv::Mat& image, **const** **int**& scale\_idx, vector<**int**>& fern){
78. **int** leaf;  //叶子  树的最终节点
79. //每一个每类器维护一个后验概率的分布，这个分布有2^d个条目（entries），这里d是像素比较pixel comparisons
80. //的个数，这里是structSize，即13个comparison，所以会产生2^13即8,192个可能的code，每一个code对应一个后验概率
81. **for** (**int** t=0; t<nstructs; t++){  //nstructs 表示树的个数 10
82. leaf=0;
83. **for** (**int** f=0; f<structSize; f++){  //表示每棵树特征的个数 13
84. //struct Feature 特征结构体有一个运算符重载 bool operator ()(const cv::Mat& patch) const
85. //返回的patch图像片在(y1,x1)和(y2, x2)点的像素比较值，返回0或者1
86. //然后leaf就记录了这13位的二进制代码，作为特征
87. leaf = (leaf << 1) + features[scale\_idx][t\*nstructs+f](image);
88. }
89. fern[t] = leaf;
90. }
91. }
93. **float** FerNNClassifier::measure\_forest(vector<**int**> fern) {
94. **float** votes = 0;
95. **for** (**int** i = 0; i < nstructs; i++) {
96. // 后验概率posteriors[i][idx] = ((float)(pCounter[i][idx]))/(pCounter[i][idx] + nCounter[i][idx]);
97. votes += posteriors[i][fern[i]];   //每棵树的每个特征值对应的后验概率累加值 作投票值？？
98. }
99. **return** votes;
100. }
102. //更新正负样本数，同时更新后验概率
103. **void** FerNNClassifier::update(**const** vector<**int**>& fern, **int** C, **int** N) {
104. **int** idx;
105. **for** (**int** i = 0; i < nstructs; i++) {
106. idx = fern[i];
107. (C==1) ? pCounter[i][idx] += N : nCounter[i][idx] += N;
108. **if** (pCounter[i][idx]==0) {
109. posteriors[i][idx] = 0;
110. } **else** {
111. posteriors[i][idx] = ((**float**)(pCounter[i][idx]))/(pCounter[i][idx] + nCounter[i][idx]);
112. }
113. }
114. }
116. //训练集合分类器（n个基本分类器集合）
117. **void** FerNNClassifier::trainF(**const** vector<std::pair<vector<**int**>,**int**> >& ferns,**int** resample){
118. // Conf = function(2,X,Y,Margin,Bootstrap,Idx)
119. //                 0 1 2 3      4         5
120. //  double \*X     = mxGetPr(prhs[1]); -> ferns[i].first
121. //  int numX      = mxGetN(prhs[1]);  -> ferns.size()
122. //  double \*Y     = mxGetPr(prhs[2]); ->ferns[i].second
123. //  double thrP   = \*mxGetPr(prhs[3]) \* nTREES; ->threshold\*nstructs
124. //  int bootstrap = (int) \*mxGetPr(prhs[4]); ->resample
126. //thr\_fern: 0.6 thrP定义为Positive thershold
127. thrP = thr\_fern \* nstructs;                                    // int step = numX / 10;
128. //for (int j = 0; j < resample; j++) {                      // for (int j = 0; j < bootstrap; j++) {
129. **for** (**int** i = 0; i < ferns.size(); i++){               //   for (int i = 0; i < step; i++) {
130. //     for (int k = 0; k < 10; k++) {
131. //       int I = k\*step + i;//box index
132. //       double \*x = X+nTREES\*I; //tree index
133. **if**(ferns[i].second==1){    //为1表示正样本        //       if (Y[I] == 1) {
134. //measure\_forest函数返回所有树的所有特征值对应的后验概率累加值
135. //该累加值如果小于正样本阈值，也就是是输入的是正样本，却被分类成负样本了
136. //出现分类错误，所以就把该样本添加到正样本库，同时更新后验概率
137. **if**(measure\_forest(ferns[i].first) <= thrP)      //         if (measure\_forest(x) <= thrP)
138. ////更新正样本数，同时更新后验概率
139. update(ferns[i].first, 1, 1);                 //             update(x,1,1);
140. }**else**{                                            //        }else{
141. **if** (measure\_forest(ferns[i].first) >= thrN)   //         if (measure\_forest(x) >= thrN)
142. update(ferns[i].first, 0, 1);                 //             update(x,0,1);
143. }
144. }
145. //}
146. }
148. //训练最近邻分类器
149. **void** FerNNClassifier::trainNN(**const** vector<cv::Mat>& nn\_examples){
150. **float** conf, dummy;
151. vector<**int**> y(nn\_examples.size(),0); //vector<T> v3(n, i); v3包含n个值为i的元素。y数组元素初始化为0
152. y[0]=1;  //上面说到调用trainNN这个函数传入的nn\_data样本集，只有一个pEx，在nn\_data[0]
153. vector<**int**> isin;
154. **for** (**int** i=0; i<nn\_examples.size(); i++){                          //  For each example
155. //计算输入图像片与在线模型之间的相关相似度conf
156. NNConf(nn\_examples[i], isin, conf, dummy);                      //  Measure Relative similarity
157. //thr\_nn: 0.65 阈值
158. //标签是正样本，如果相关相似度小于0.65 ，则认为其不含有前景目标，也就是分类错误了；这时候就把它加到正样本库
159. **if** (y[i]==1 && conf <= thr\_nn){                                //    if y(i) == 1 && conf1 <= tld.model.thr\_nn % 0.65
160. **if** (isin[1]<0){                                          //      if isnan(isin(2))
161. pEx = vector<Mat>(1,nn\_examples[i]);                 //        tld.pex = x(:,i);
162. **continue**;                                            //        continue;
163. }                                                        //      end
164. //pEx.insert(pEx.begin()+isin[1],nn\_examples[i]);        //      tld.pex = [tld.pex(:,1:isin(2)) x(:,i) tld.pex(:,isin(2)+1:end)]; % add to model
165. pEx.push\_back(nn\_examples[i]);
166. }                                                            //    end
167. **if**(y[i]==0 && conf>0.5)                                      //  if y(i) == 0 && conf1 > 0.5
168. nEx.push\_back(nn\_examples[i]);                             //    tld.nex = [tld.nex x(:,i)];
170. }                                                                 //  end
171. acum++;
172. printf("%d. Trained NN examples: %d positive %d negative\n",acum,(**int**)pEx.size(),(**int**)nEx.size());
173. }                                                                  //  end
175. /\*Inputs:
176. \* -NN Patch
177. \* Outputs:
178. \* -Relative Similarity (rsconf)相关相似度, Conservative Similarity (csconf)保守相似度,
179. \* In pos. set|Id pos set|In neg. set (isin)
180. \*/
181. **void** FerNNClassifier::NNConf(**const** Mat& example, vector<**int**>& isin,**float**& rsconf,**float**& csconf){
182. isin=vector<**int**>(3,-1);  //vector<T> v3(n, i); v3包含n个值为i的元素。 三个元素都是-1
183. **if** (pEx.empty()){ //if isempty(tld.pex) % IF positive examples in the model are not defined THEN everything is negative
184. rsconf = 0; //    conf1 = zeros(1,size(x,2));
185. csconf=0;
186. **return**;
187. }
188. **if** (nEx.empty()){ //if isempty(tld.nex) % IF negative examples in the model are not defined THEN everything is positive
189. rsconf = 1;   //    conf1 = ones(1,size(x,2));
190. csconf=1;
191. **return**;
192. }
193. Mat ncc(1,1,CV\_32F);
194. **float** nccP, csmaxP, maxP=0;
195. **bool** anyP=**false**;
196. **int** maxPidx, validatedPart = ceil(pEx.size()\*valid);  //ceil返回大于或者等于指定表达式的最小整数
197. **float** nccN, maxN=0;
198. **bool** anyN=**false**;
199. //比较图像片p到在线模型M的距离（相似度），计算正样本最近邻相似度，也就是将输入的图像片与
200. //在线模型中所有的图像片进行匹配，找出最相似的那个图像片，也就是相似度的最大值
201. **for** (**int** i=0;i<pEx.size();i++){
202. matchTemplate(pEx[i], example, ncc, CV\_TM\_CCORR\_NORMED);      // measure NCC to positive examples
203. nccP=(((**float**\*)ncc.data)[0]+1)\*0.5;  //计算匹配相似度
204. **if** (nccP>ncc\_thesame)  //ncc\_thesame: 0.95
205. anyP=**true**;
206. **if**(nccP > maxP){
207. maxP=nccP;    //记录最大的相似度以及对应的图像片index索引值
208. maxPidx = i;
209. **if**(i<validatedPart)
210. csmaxP=maxP;
211. }
212. }
213. //计算负样本最近邻相似度
214. **for** (**int** i=0;i<nEx.size();i++){
215. matchTemplate(nEx[i],example,ncc,CV\_TM\_CCORR\_NORMED);     //measure NCC to negative examples
216. nccN=(((**float**\*)ncc.data)[0]+1)\*0.5;
217. **if** (nccN>ncc\_thesame)
218. anyN=**true**;
219. **if**(nccN > maxN)
220. maxN=nccN;
221. }
222. //set isin
223. //if he query patch is highly correlated with any positive patch in the model then it is considered to be one of them
224. **if** (anyP) isin[0]=1;
225. isin[1]=maxPidx;      //get the index of the maximall correlated positive patch
226. //if  the query patch is highly correlated with any negative patch in the model then it is considered to be one of them
227. **if** (anyN) isin[2]=1;
229. //Measure Relative Similarity
230. //相关相似度 = 正样本最近邻相似度 / （正样本最近邻相似度 + 负样本最近邻相似度）
231. **float** dN=1-maxN;
232. **float** dP=1-maxP;
233. rsconf = (**float**)dN/(dN+dP);
235. //Measure Conservative Similarity
236. dP = 1 - csmaxP;
237. csconf =(**float**)dN / (dN + dP);
238. }
240. **void** FerNNClassifier::evaluateTh(**const** vector<pair<vector<**int**>,**int**> >& nXT, **const** vector<cv::Mat>& nExT){
241. **float** fconf;
242. **for** (**int** i=0;i<nXT.size();i++){
243. //所有基本分类器的后验概率的平均值如果大于thr\_fern，则认为含有前景目标
244. //measure\_forest返回的是所有后验概率的累加和，nstructs 为树的个数，也就是基本分类器的数目 ？？
245. fconf = (**float**) measure\_forest(nXT[i].first)/nstructs;
246. **if** (fconf>thr\_fern)  //thr\_fern: 0.6 thrP定义为Positive thershold
247. thr\_fern = fconf;  //取这个平均值作为 该集合分类器的 新的阈值，这就是训练？？
248. }
250. vector <**int**> isin;
251. **float** conf, dummy;
252. **for** (**int** i=0; i<nExT.size(); i++){
253. NNConf(nExT[i], isin, conf, dummy);
254. **if** (conf > thr\_nn)
255. thr\_nn = conf; //取这个最大相关相似度作为 该最近邻分类器的 新的阈值，这就是训练？？
256. }
258. **if** (thr\_nn > thr\_nn\_valid)  //thr\_nn\_valid: 0.7
259. thr\_nn\_valid = thr\_nn;
260. }
262. //把正样本库（在线模型）包含的所有正样本显示在窗口上
263. **void** FerNNClassifier::show(){
264. Mat examples((**int**)pEx.size()\*pEx[0].rows, pEx[0].cols, CV\_8U);
265. **double** minval;
266. Mat ex(pEx[0].rows, pEx[0].cols, pEx[0].type());
267. **for** (**int** i=0;i<pEx.size();i++){
268. //minMaxLoc寻找矩阵（一维数组当作向量，用Mat定义）中最小值和最大值的位置.
269. minMaxLoc(pEx[i], &minval); //寻找pEx[i]的最小值
270. pEx[i].copyTo(ex);
271. ex = ex - minval;  //把像素亮度最小的像素重设为0，其他像素按此重设
272. //Mat Mat::rowRange(int startrow, int endrow) const 为指定的行span创建一个新的矩阵头。
273. //Mat Mat::rowRange(const Range& r) const   //Range 结构包含着起始和终止的索引值。
274. Mat tmp = examples.rowRange(Range(i\*pEx[i].rows, (i+1)\*pEx[i].rows));
275. ex.convertTo(tmp, CV\_8U);
276. }
277. imshow("Examples", examples);
278. }

# [庖丁解牛TLD](http://blog.csdn.net/yang_xian521/article/details/6952870)

## 开篇

最近在网上多次看到有关Zdenek Kalal的TLD的文章，说他做的工作如何的帅，看了一下TLD的视频，感觉确实做的很好，有人夸张的说他这个系统可以和Kniect媲美，我倒是两者的工作可比性不大，实现的方法也不同。但这个哥们做的真的很棒，最可贵的是人家提供了源码可以下载。他相关的工作网上一搜一大片，推荐一个链接<http://www.cvchina.net/article-22-1.html>，再给个作者网站的链接<http://info.ee.surrey.ac.uk/Personal/Z.Kalal/>。



看了他的demo视频顿时心潮澎湃，赶紧下载了他的源代码后又有些崩溃了。他的工作量还是很大的，得静下心来慢慢研究。直接看代码难免云里雾里的，先看看人家的论文吧。

这哥们这几年可真没少发paper啊，我比较关注他实现的算法。TLD即Tracking Learning Detector，我认为就是依据跟踪学习的目标检测。看了他08年发表的一篇文章，介绍了他的这个算法，那时候还把算法称为TMD（tracking、modeling、detection），他的tracking工作时基于Lucas-Kanade光流法的。modeling学习的过程有growing和pruning两方面的工作，可以正负反馈，得到较好的学习结果，对于他的学习过程，他在另一篇文章中又详细介绍了P-N learning这种学习算法。detection的部分用的是随机森林的机器学习办法，加上bootstarps。对于特征的选择，他提出了一种基于LBP特征的2bit BP特征。几篇文章下来，我已经有点晕乎了，这些算法都不是太熟悉，看来得结合代码一个个啃了。

代码正在一点点啃，有同样兴趣的朋友可以交流指导一下，不胜感激。

## [初始化工作（为算法的准备）](http://blog.csdn.net/yang_xian521/article/details/6957139)

我说的初始化，还不是算法的初始化工作，而是读入图像，响应键盘鼠标之类的工作。作者提供的代码中的工作包含了从摄像头读取和从文件中读取两种输入方案。这里介绍一下从文件输入的办法。因为OpenCV从视频读取图像序列的办法有很好的demo，我这里就不介绍摄像头的办法了。TLD下载后有一个文件夹是\_input，里面存放着一组图片组，图片文件的名字为00001.png、00002.png....。我读取图片组的关键代码如下，这段代码具有普遍意义，可以移植到以后想读入图片组的任意程序中：

**[cpp]** [view plaincopy](http://blog.csdn.net/yang_xian521/article/details/6957139)

1. stringstream fileNameStream;
2. string sourceImageFileName;
3. for(**int** nFrame=0; nFrame<NUM; nFrame++)
4. {
5. nFrame++;
6. fileNameStream << "\_input\\" << setw(5) << setfill('0') << nFrameNum << ".png";
7. sourceImageFileName = fileNameStream.str();
8. fileNameStream.clear();
9. fileNameStream.str("");
11. // 读取图像
12. g\_src = imread(sourceImageFileName);
13. }

这就实现了图片的读入工作，再参考camshiftdemo的办法实现了鼠标和键盘的响应。鼠标的响应就是得到目标区域的范围，用鼠标选中boundingbox。文件读进来了，目标区域boundingbox也得到了，接下来就是需要对算法进行研究了。

先介绍几个我研究过的心得，bbox文件夹下面的代码主要都是对boundingbox的处理。tld文件夹下面的存放的是主干的算法，从run\_TLD入手，感觉就是对起始帧进行初始化工作，然后逐帧读入图片序列，进行算法处理。还是先分析初始化工作，作者的tldInitSource函数实现的就是基本的初始化，给一些变量赋值，开辟矩阵大小，这个没什么好讲的。tldInitFirstFrame文件完成的工作就是选中boundingbox，这个功能我已经通过鼠标的响应得到了boundingbox，也可以略过不细分析。重点的初始化工作是在tldInit里实现的，这个函数也是我接下来研究的重点，本人Matlab较差，真希望有高人指点啊，一起研究啊

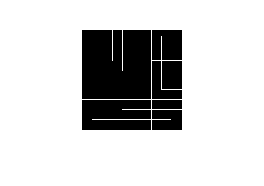
## [算法初始化](http://blog.csdn.net/yang_xian521/article/details/6957732)

上一讲我提到对于算法的初始化工作主要是在tldInit这个函数里实现的。主要分为如下几大步骤，1）初始化Detector。2）初始化Trajectory。3）训练Detector

### 1)初始化Detector

其中bb\_scan为扫描grid区域，该函数输入为boundingBox，输出为一系列的RectBox，是根据boundingBox的大小参数对待搜素区域选择一系列的box作为备选的跟踪区域，box的位置和尺度都有变化，和RectBox相应的尺度。但RectBox有6个参数，前4个分别为Rect的左上角坐标（x1，y1）和右下角坐标（x2，y2）。后两个参数求大神解释（PS：后来在fern函数里找到了解释，分别为指向对应尺度特征的指针位置、每一行box的数量——用在搜索邻近box）！对于这个函数内部我还有一个疑惑，就是对ntuples函数功能的使用，哎，怎奈Matlab语法都不熟悉，只能慢慢啃了，感觉作者这里就是把RectBox的左上角的所有可能的坐标值传入该函数，得到左上角坐标位置的全部组合（不知道理解对了没）。

接下来的工作时特征的初始化，是在tldGenerateFeature函数里实现的。这个函数相对独立，作者这里为了产生效果较好的随机特征真是煞费苦心，输入的参数有两个，一个是nTREE = 10，一个是nFEATURE = 13。输出为nTREE组特征，每组特征为nFEATURE个点对，每一个点对有4个参数，分别两点坐标（x1，y1），（x2，y2），取值范围为（0~1）其中第一个点的分辨率为0.1，还不太明白这样设计的原因，待进一步分析代码，有高人指点一下更好。值得注意的是产生的点对不是横坐标相同x1 = x2，就是纵坐标相同y1 = y2。这里用图片显示一组特征，线段的两个端点



下一步工作为初始化detector。这个功能是用强大的fern函数写的，该函数有多个功能，根据传入参数的标志分别可以实现clear操作、init操作、update操作、evaluate操作、detect操作、get pattern操作。fern函数是用c写的，混合编程没有弄的太明白，还没能调试一下看看，只能看代码猜。在初始化detector的工作里，用到的是init操作。

### 2）初始化Trajectory

这部分没有什么要说明的，都是些零碎的初始化工作，matlab里面对一些必要的变量开辟一些空间和定义一些变量的值。具体分析Trajectory的工作的时候可以具体再分析

### 3）训练Detector

首先得到Target，作者注释说该Target只是用来显示，有待我后续验证。得到Target要调用函数img\_patch，img\_patch函数是获得一幅图像中目标区域box的像素信息patch。

接下来产生正样本数据集，调用tldGeneratePositiveData。其中第二个参数为RectBox和box的重复区域比例信息，保存在overlap参数中，由函数bb\_overlap得到。tldGeneratePositiveData函数首先根据overlap的比例信息选出重复区域比例大于60%并且前num\_closet  = 10个的最接近box的RectBox，相当于对RectBox进行筛选。并通过bb\_hull函数得到这些RectBox的最大边界。接下来的工作比较重要，要得到Pattern，调用的函数为tldGetPattern。初始化的工作就是对最接近box的RectBox区域得到其patch，然后调用tldPatch2Pattern将像素信息转换为Pattern，具体的说就是归一化RectBox对应的patch的size（放缩至patch\_size = 15\*15），将2维的矩阵变成一维的向量信息，然后将向量信息均值设为0，调整为zero mean and unit variance（ZMUV），这个过程调用函数tldPatch2Pattern实现。接下来处理RectBox最大边界的模糊信息，再次用到img\_patch函数，但这次调用的函数有很大的不同，还没太理解作者要做什么啊，怎么感觉还有平移旋转矩阵都出来了，晕啦（求高人指点）。该函数最后返回3个参数，pX为处理后的RectBox最大边界处理后的像素信息，pEx最近邻的RectBox的Pattern，bbP0为最近邻的RectBox。

然后再产生负样本数据tldGenerateNegativeData。得到远离box（重复区域比例小于20%）的num\_patches = 100个Pattern保存到nX中，随机选中num\_patches = 100个RectBox得到对应的patch保存到nEx中。这里调用了fern（5），即该函数的get pattern操作。

接下来对负样本进行分类，分类到训练集Training Set和验证集Validation Set中去。

接下来使用Training Set进行训练，先调用fern（2），更新，然后调用tldTrainNN最近邻训练数据。

接下来评估验证集Validation Set的阈值。调用tldNN验证。

至此，初始化的工作基本完成，限于本人水平有限，只能先对函数有个大概的认识，深深觉得先要静下心来把训练的算法搞清楚，再回过头来再看一编代码。

## [Tracking解析](http://blog.csdn.net/yang_xian521/article/details/7079749)

前几节都是根据作者的程序流程一步步介绍作者的工作，感觉只是对代码的一个注释，这次换一个思路，一部分一部分啃，作者的工作主要就是3部分么，tracking，learning，detection。

这次先介绍Tracking的工作。对于Tracking，作者主要使用的是他提出的Forward-Backward Error的办法，使用Lucas-Kanade光流法跟踪，对跟踪的结果，用Forward-Backward Error做反馈，求FB error的结果与原始位置的欧式距离，把距离过大的跟踪结果舍弃，他把这种利用FB error舍弃坏值的跟踪方法叫做Median Flow，是把欧式距离集合中较大的50%的那些跟踪结果舍弃。作者在他的文章Forward-Backward Error：Automatic Detection of Tracking Failures里提到用FB+NCC（交叉验证）的方案，可以使跟踪的结果最佳。作者的Tracking的办法就是根据我以上介绍的流程实现的。接下来结合代码再详细剖析一下

先用bb\_points函数在box中均匀采样10\*10个点，注意作者这里设置了采样点的区域比box的区域少一圈边界，边界为5，在后面我会介绍作者这里的独到用心。然后调用混合编程的lk函数实现lucas-Kanade光流法跟踪，得到的结果有为这100个点的lk结果，前两个参数为利用l-k方法得到的点当前的跟踪位置坐标，第三个参数是利用NCC把跟踪预测的结果周围取10\*10的小图片与原始位置周围10\*10（这里取10\*10，有心的朋友应该笑了，为什么作者之前在bb\_points函数里要设置个边界5，原来是防止越界哦）的小图片（使用函数getRectSubPix得到）进行模板匹配（调用matchTemplate），再对匹配的结果归一化，把这个结果保存在第三个参数中，第四个参数为FB error的欧氏距离。这个lk函数过程中有很多参数可以设置，对最终的结果我想应该应该也是有的，有待实验验证。接下来就是利用作者提出的Median Flow，得到NCC和FB error结果的中值，分别去掉中值一半的跟踪结果不好的点，利用这一半（其实不到50%）的跟踪点输入函数bb\_predict函数中预测bounding box在当前帧的位置和大小。

这基本就是Tracking工作的主要部分了，至于被遮挡的tracking（tldTrack\_occlusion），作者进行了单独处理，下一次再分析。

PS：很感谢最近有些网友与我一起研究TLD，不过本人能力不足，很多东西还是不理解，对于作者detection和learning的工作，感觉那部分的代码实在好比天书，没法拿出来和大家交流了，希望有识之士也能写出来，和大家分享~~

## [井底之蛙啦~](http://blog.csdn.net/yang_xian521/article/details/7091587)

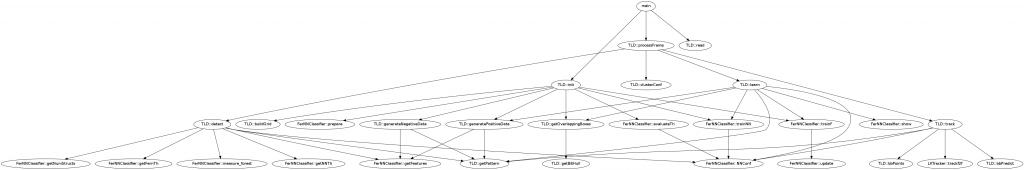
随着和我交流TLD的朋友越来越多，我渐渐的知道的也多了，才发现我研究的结果只是沧海一粟。

这里先膜拜一下Alan Torres大神，他已经用c++把TLD重新写好了，而且代码很规范。他设计的理念有：

1. depends \*only\* on OpenCV (2.3)   
2. no Matlab!   
3. easy to compile and run (on linux, work in progress on OSX and windows)   
4. fast! (and more potential to be much faster)   
5. No Matlab! (did I say no matlab?)

没有matlab，多平台，更快的速度。真好，就是我想做的，不过他现在这个程序，在我这电脑上实现速度还不行。他代码的下载地址为[https://github.com/alantrrs/OpenTLD](http://www.google.com/url?sa=D&q=https://github.com/alantrrs/OpenTLD&usg=AFQjCNFU8PdcI0cQ79FR5TF6Lk04D6GePw)，好像打不开，我是在这上面得到的[https://github.com/arthurv/OpenTLD](http://www.google.com/url?sa=D&q=https://github.com/arthurv/OpenTLD&usg=AFQjCNGDXLUFLgmCuf4Np-Tr6241h_G6fA)。不过是个Linux版本的。大家这么强，改改肯定就可以在xp下跑起来了，反正我是搞定了。

附上他软件的设计接口。真是赏心悦目啊，不得不说人家做的东西很规范，惭愧惭愧



看不清还是下载下来大家自己看咯，不好意思，我不知道怎么能传上去看得清晰

从他这个设计图也可以看出来我之前几讲分析的流程还是可以接受的，init部分和track部分是相对独立的。而比较复杂的是learning的部分和detect部分。下一步主攻这两部分了。学习的越深入，越是发现自己很挫，都没信心继续写下去了。硬着头皮装大葱吧~

# TLD源码深度分析：

## 初始化模块

            对于TLD的研究，已有些时日。最开始的时候看Zdenek Kalal的论文，但是有很多啃不动的地方。看完只觉得了解了个大体框架，很多地方都不明白。后来，又看完了各位牛人的分析及讲解（大部分已转载到本博客，大家可以查阅），受益匪浅！但是觉得对很多问题还是不明白！好在网友【冷月无心】给我推荐了C++版本作者的论文，作者分析得很细，看完很受用，解除了很多困惑！最近，打算再根据C++源码深入地走一遍，并把源码功能详细分享给大家！希望后面学习使用TLD算法的朋友通过我的分析，能够不走那么多弯路，也欢迎朋友们和我一起讨论！废话就说到这里，开始分享。。。

**1.**       **buildGrid(frame1，box)**

输入：

      当前帧图像：frame1

      初始边界框：box

输出：

          所有边界框及其信息：vector<BoundingBox> grid

    描述：

使用滑动窗法对整帧图像，按从上到下、从左到右的顺序，取边界框并记录其大小、位置、尺度、与初始边界框（第一帧中手工所取）重叠度。其中，尺度取初

始边界框的1.2(S次方)倍，S取值为-10、-9、…、9、10。

**2.**       **getOverlappingBoxes(box,num\_closest\_init)**

输入：

      初始边界框：box

      欲得到good\_box(与初始边界框最相似的box)的个数：num\_closest\_init

输出：

      最相似边界框：best\_box

         最相似的num\_closest\_init个边界框：vector<int> good\_boxes

          不重叠边界框：vector<int> bad\_boxes

描述：

将所有边界框中，与初始框box重叠度最高的边界框信息赋给best\_box。把所有重叠度高于0.6的边界框归类到good\_boxes;把所有重叠度低于阈值bad\_overlap（读取自parameters.yml，论文中为0.2）的边界框归类到bad\_boxes。如果good\_boxes中边界框个数多于num\_closest\_init个，则从中取重叠率最高num\_closest\_init个边界框good\_boxes。函数末尾调用getBBHull()函数的目的是，得到good\_boxes中所有边界框能覆盖到的最大边界。

**3.**       **classifier.prepare(scales)：**函数位于FerNNClassifier.cpp中。

输入：

所有有效尺度：scales

   输出：

         所有待获取2bitBP特征（位置）：features

         阈值：0.5\*nstructs

         初始化Posteriors、pCounter、nCounter

   描述：

对特征提取的位置（均匀任选图像块中两点）进行选取（共计：蕨个数\*特征位数

（二进制数位数）\*尺度大小（21） 个），声明并初始化每个蕨的可信度、正样

本计数器、负样本计数器。

**4.**       **generatePositiveData(frame1,num\_warps\_init)**

输入：

当前帧图像：frame1

图像块变换次数：num\_warps\_init

最相似边界框：best\_box

相似边界框：good\_boxes

   输出：

              集成分类器正样本集：pX

              best\_box图像块处理结果：pEx

描述：

getPattern(frame(best\_box),pEx,mean,stdev):

frame(best\_box)为该帧图像中best\_box边界框对应图像块；将该图像块resize

为15\*15的大小；计算该图像的均值、方差，分别存放到mean、stdev中；将该

图像块与均值作差的结果存于pEx中。

        generator(frame,pt,warped,bbhull.size(),rng)：

        pt为good\_boxes最大边界框（**2**中有提到）的中心点；该函数对每个good\_box对

应图像块仿射变换：±1%范围的偏移，±1%范围的尺度变化，±10%范围的平面

内旋转，并且在每个像素上增加方差为5的高斯噪声（确切的大小是在指定的范

围内随机选择的），共计num\_warps\_init种。

**关于该部分，详情见**[**《关于PatchGenerator类 》**](http://quandb2007.blog.163.com/blog/static/4187887520135903115980/)

     对每一仿射图像块计算特征，并存于pX中。

        classifier.getFeatures(patch,grid[idx].sidx,fern)：

        对尺度grid[idx].sidx下的图像块patch，使用**3**中定义的位置计算特征，并存于

        fern中。

**注意：**

         原C++版本作者此处代码有误。没有用到变换后的图像块。

   （也有朋友说没有错，待验证，但修改的一定是正确的）

修改方法一：

    将generator(frame,pt,warped,bbhull.size(),rng);

    修改成generator(img,pt,img,frame.size(),rng);

修改方法二：

    将整个for循环内容修改为：

    for (int i=0;i<num\_warps;i++)

   {

     if(i == 0)

     {

for (int b = 0; b < good\_boxes.size(); b ++)

{

   idx=good\_boxes[b];

   patch = img(grid[idx]);

   classifier.getFeatures(patch, grid[idx].sidx, fern);

   pX.push\_back(make\_pair(fern,1));

}

   }

  else

   {

      generator(img,pt,warped,bbhull.size(),rng);

      for (int b = 0; b < good\_boxes.size(); b ++)

      {

idx=good\_boxes[b];

Rect region(grid[idx].x-bbhull.x, grid[idx].y - bbhull.y, grid[idx].width, grid[idx].height);

patch = warped(region);

classifier.getFeatures(patch, grid[idx].sidx, fern);

pX.push\_back(make\_pair(fern,1));

     }

  }

}

**5.**       **meanStdDev(frame1(best\_box),mean,stdev)**

输入：

最相似图像块：frame1(best\_box)

输出：

图像块均值：mean

图像块标准差：stdev

描述：

计算best\_box边界框对应图像块的均值和标准差，并分别存于mean、stdev中。

**6.**       **integral(frame1,iisum,iisqsum)**

输入：

        整帧图像：frame1

输出：

        积分图：iisum

        平方积分图：iisqum

描述：

        对整帧图像计算积分图、图像平方的积分图

**7.**       **generateNegativeData(frame1)**

输入：

         整帧图像：frame1

              模板个数：bad\_patches

        所有不重叠边界框：bad\_boxes

        积分图：iisum

        平方积分图：iisqum

输出：

         负样本特征集：nX

         负样本模板集：nEx

描述：

               random\_shuffle(bad\_boxes.begin(),bad\_boxes.end())：

     随机打乱bad\_boxes的顺序。取所有bad\_boxes边界框中，方差值高于best\_box图像块阈值一半的图像块，计算特征并保存到nX中。(why?：因为检测的时候，方差分类器位于集合分类器的前一级。也就是说，方差低于0.5倍best\_box图像块方差的部分都被排除，如果在此处将其作为负样本是没有意义的，甚至会降低集合分类器的检测效果。)  任取bad\_patches个图像块作为负模板。

注意：

      原程序版本有错，对方差分类器方差重复取50%。

      将if (getVar(grid[idx],iisum,iisqsum)<var\*0.5f)

      修改成if (getVar(grid[idx],iisum,iisqsum)<var\*1.0f)

**8.**       **classifier.trainF(ferns\_data,2)**

输入：

          已混合打乱的正负样本特征数据：ferns\_data

输出：

        分类器模型：任何特征对应可信度(该图像块为正样本的概率)：posteriors

描述：

     统计P/N约束被使用的次数，来计算一个特征的可信度

更多详情见[《集成分类器》](http://quandb2007.blog.163.com/blog/static/4187887520135943158981/)

**9.**       **classifier.trainNN(nn\_data)**

输入：

     已混合打乱的正负样本模板数据：nn\_data

输出：

     筛选出的模板：pEx、nEx

描述：

     通过计算自相关系数NCC筛选模板

更多详情见[《最近邻分类器》](http://quandb2007.blog.163.com/blog/static/418788752013597292355/)

**10.**       **classifier.evaluateTh(nXT,nExT)**

输入：

         负样本随机蕨测试集：nXT

         负样本模板测试集：nExT

输出：

            更新后的随机蕨阈值：thr\_fern

            更新后的模板NCC阈值：thr\_nn

            更新后的有效性判断(P/N学习)阈值：thr\_nn\_valid

   描述：

             对所有负样本随机蕨测试集，计算其随机蕨可信度，若最大值大于预设阈值

             thr\_fern，则用该最大值替换更新；对所有负样本模板测试集，计算其NCC系数，

      若最大值大于预设阈值thr\_nn，则用该最大值替换更新；若最后更新得到的thr\_nn

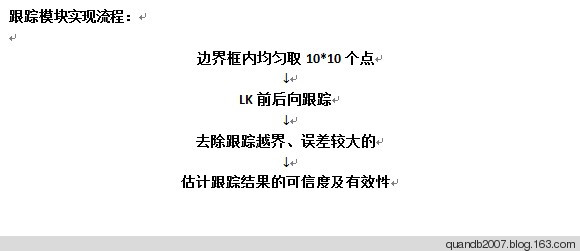
大于预设thr\_nn\_valid，则替换更新thr\_nn\_valid。

注：若存在说法看不明白的，请留言询问！

      至此，TLD初始化模块介绍完毕！对整个模块的梳理，见[《TLD初始化模块实现流程》](http://quandb2007.blog.163.com/blog/static/418788752013593451825/)

## 跟踪模块

**跟踪模块track(img1,img2,points1,points2)**

[](http://quandb2007.blog.163.com/)

输入：

         上一帧图像：img1

         当前帧图像：img2

        前一帧边界框：lastbox

输出：

      跟踪结果边界框

      其可信度及有效性

**1.**       **bbPoints(points1,lastbox)**

输入：

边界框：lastbox

   输出：

              点集：points1

   描述：

             在边界框lastbox中均匀采集10\*10个点坐标，存于points1中。

**2.**       **tracked = tracker.trackf2f(img1,img2,points,points2)**

输入：

               上一帧图像：img1

              当前帧图像：img2

              前一帧边界框中的初始点：points(1中的points1)

   输出：

             得到前一帧中被有效跟踪的点集：points

             得到对当前帧有效跟踪点的点集：points2

描述：

               首先，使用LK光流法进行前向跟踪（calcOpticalFlowPyrLK( img1,img2, points1,

   points2, status,similarity, window\_size, level, term\_criteria, lambda, 0);），

   再进行后向跟踪（calcOpticalFlowPyrLK( img2,img1, points2, pointsFB, FB\_status,

   FB\_error, window\_size, level, term\_criteria, lambda, 0);）；然后，计算跟踪误差

 （FB\_error[i] = norm(pointsFB[i]-points1[i])）；最后，计算跟踪点与初始点周围

   10\*10像素块的归一化互相关系数，滤除掉相似性低于均值的点，在滤除前后向误差

  高于均值的点，得到最终有效跟踪点。

**3.**       **bbPredict(points,points2,lastbox,tbb)**

输入：

         前一帧中被有效跟踪的点：points

        当前帧成功跟踪到的点：points2

        前一帧中初始（或有效最终结果）边界框：last\_box

输出：

        跟踪结果边界框：tbb

描述：

根据这些成对对应点，计算它们位移变化的均值；根据原边界框内的点两两间距

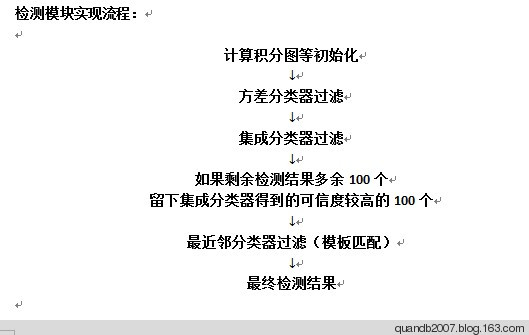
离，与跟踪到的对应两个点间的距离，二者比值关系的均值，得到前后两帧中目标的放缩变化。再根据前一边界框的大小及位置，预测当前跟踪框的大小及位置。

 如果前后向误差均值大于10，或预测得到的边界框越界，则本次跟踪失败，得到跟踪结果无效。

## 检测模块

结合初始化模块的函数解释，本模块代码能够很容易就看明白！（还有不明白的，请留言或者邮件咨询！）

故针对此模块，只给出其实现流程：

[](http://quandb2007.blog.163.com/)

## 追踪、检测结果归并模块

**合并模块**

**1.**  **clusterConf(dbb,dconf,cbb,cconf)**

输入：

          检测到的边界框集：dbb

          检测结果可信度集：dconf

输出：

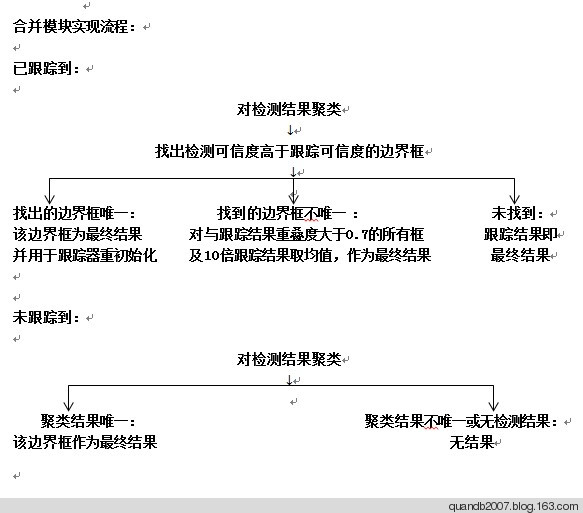
      聚类后的边界框集：cbb

      聚类后的可信度集：cconf

描述：

        使用partition(dbb,T,(\*bbcomp))对边界框集dbb进行聚类，然后将每一类中

        的框进行取均值融合成一个框。

[](http://quandb2007.blog.163.com/)

## 学习模块

**学习模块learn(img2)**

**1.**       **grid[i].overlap = bbOverlap(lastbox,grid[i])**

输入：

      最终有效边界框：lastbox

      初始化时得到的边界框集：grid

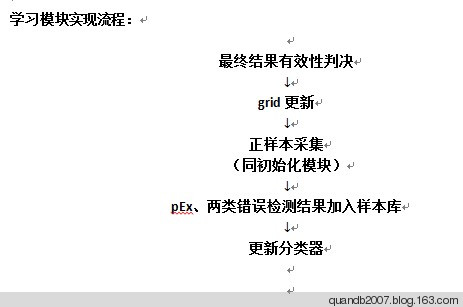
输出：

      更新后的边界框集：grid

描述：

      本函数的主要目的是，针对初始的grid而言，新的初始化目标边界框已改变，故而其grid中的框与目标边界框的重叠度已经发生改变，需要重新计算更新。

注：本模块其它所用函数在初始化模块已有说明，此处不再累述。

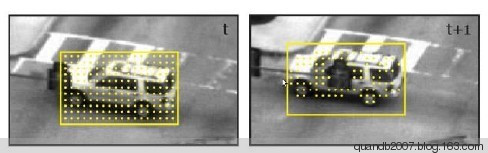
[](http://quandb2007.blog.163.com/)

**说明：关于两类错误**

一、可信度大于等于阈值的bad\_box，加入集成分类器样本库；【作者此处取的阈值为1，导致好些朋友不理解（可信度最高才1，咋可能大于？）】

二、被检测为正，但其重叠度小于阈值bad\_overlap的图像块，加入最近邻分类器样本库。

## 关于TLD算法的跟踪模块的补充说明

[](http://quandb2007.blog.163.com/blog/static/41878875201351241741534/)

         首先，在边界框中均匀取10\*10个点（由于计算两点之间的距离即步长时，要用整数，所以实际取得的点数可能不是10\*10个）Points1，如上图左图所示；

[](http://quandb2007.blog.163.com/blog/static/41878875201351241741534/)

        然后， 使用光流，通过上一帧图像，预测出那些点在当前帧中的位置Points2；

        其次，再次使用光流，通过当前帧图像及Points2，追溯出Points2对应在上一帧中的位置PointsFB；

        接下来，计算PointsFB-Points，即为前后向误差（forward-backward error）；（如果误差均值大于阈值，则认为存在太多不稳定跟踪点，从而认为跟踪失败）

        再接下来，取上一帧中以点Points1为中心，10\*10的像素块，计算它与当前帧中对应点Points2为中心10\*10像素块的相似度。

        接着，取满足 相似度大于所有相似度中值(排序)，前后向误差大于所有误差中值(排序)  的点，作为最终有效的跟踪点；

        最后，使用有效跟踪点对关系，计算尺度及形变关系，从而对上一帧的边界框做变换得到新的目标位置边界框。

## TLD初始化模块实现流程

初始化模块的目的：

                                得到，  方差分类器：方差阈值var

                                             集成分类器：每个特征对应该图像块为目标的概率

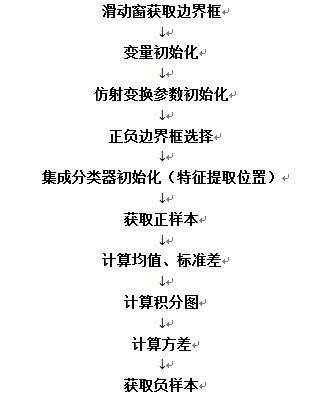
                                             最近邻分类器：模板集

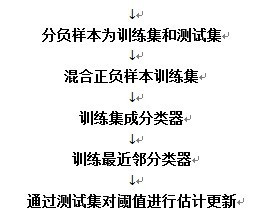
                                            随机蕨阈值：thr\_fern

                                            最近邻法判断阈值：thr\_nn

                                            有效性判决阈值：thr\_nn\_valid

完整流程：





## 随机蕨（random ferns）简介

之前看到的TLD代码中里机器学习的部分就是用的这个方法。不过TLD作者在论文中实现使用的是random forest，不过归到底，random forest和random ferns都是一种东西。

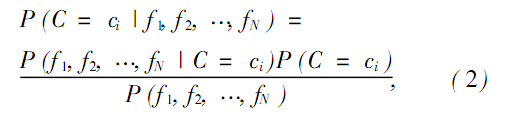
英文比较好的，推荐链接<http://cvlab.epfl.ch/alumni/oezuysal/ferns.html>，ferns方法的提出者在这里面介绍的已经很详细了。这个链接也是之前交流tld时一个网友提供的，感谢一下。

只想看中文的同志下面跟着我继续啃骨头吧，虽然我是明白了这个方法，但本人语言表述能力还有待加强啊，于是在网上搜到一篇母校的文章，以下基本摘抄自陈冰博士的paper。

ferns方法就是样本分类情况用ci，i = 1,2，。。。H表示，设fj，j = 1，2，。。。N为输入样本的二元特征集，则样本的所属类别为

http://hi.csdn.net/attachment/201203/15/0_1331816241zr1A.gif

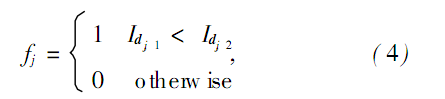
式中，C表示类的随机变量，则有



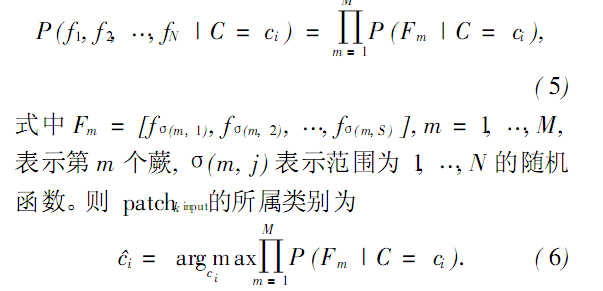
设先验概率P（C）为均匀分布，（2）式分母部分与类别无关，则（1）简化为

http://hi.csdn.net/attachment/201203/15/0_13318165305hw9.gif

这里的二元特征fj的值取决于样本中在分类器训练时随机生成的两像素位置dj1和dj2的灰度大小Idj1，Idj2比较的结果，即



由于fj的形式很简单，因此需要足够多的特征确保分类结果，但求解需要的储存量为H\*2^N字节。为降低存储量，并保证fj之间具有足够的相关性，并假设不同组的二元特征之间相互独立，组内二元特征之间具有相关性，将这些组定义为ferns feature。则（3）式的条件概率近似为



（6）式的存储量只有M\*2^S。

以上这些简单的公式都很好理解，我也很讨厌介绍东西的时候用很多公式，不过这些公式真的很简单，如果你看了这些还觉得理解的不深刻，去英文链接里看看那些图吧，会很轻松搞定的，最后还是惯例，附上ferns的代码，不过不是我写的啦，还是方法提出人的作品，老外真好，写的方法都敢拿出来和大家分享。<http://cvlab.epfl.ch/software/ferns/index.php>。

## 集成分类器

**classifier.trainF(ferns\_data,2)：**训练集成分类器

对任一随机蕨数据：对于正样本，判断其可信度是否低于阈值thr\_fern；对于负样本，判断其可信度是否大于0.5。如果满足以上条件，则使用了一次P/N约束，使用update()计数更新每个特征对应的可信度。

**update(ferns[i].first,0,1):**

void FerNNClassifier::update(const vector<int>& fern, int C, int N) {

  int idx;

  for (int i = 0; i < nstructs; i++) { //每棵树共nstructs个蕨

      idx = fern[i];//每个蕨的值（2bitBP特征组成的二进制数）

      (C==1) ? pCounter[i][idx] += N : nCounter[i][idx] += N;//计数统计

      if (pCounter[i][idx]==0) {

          posteriors[i][idx] = 0;

      } else {

          posteriors[i][idx] = ((float)(pCounter[i][idx]))/(pCounter[i][idx] + nCounter[i][idx]);

      }

  }

}

## [PN学习](http://blog.csdn.net/carson2005/article/details/7483027)

申明：以下文字为笔者阅读了计算机视觉国际会议论文之后翻译所得，其中个别部分加入了笔者自己的理解，但绝大部分都尊重了作者的原意。鉴于本人水平有限，个别地方可能存在误差，希望各位能够谅解！另外，如果您需要更详细的了解PN学习的原理、用法及应用举例，请参考原文：PN-learning:Bootstrapping Binary Classifiers by Structural Constraints, Zdenek Kalal,CVPR2010

PN学习（PN learning）是一种利用带标签的样本（一般用于分类器训练，以下均称之为测试样本）和不带标签的样本（一般用于分类器测试，以下均称之为测试样本）之间存在的结构性特征（见下面的解释）来逐步（学习）训练两类分类器并改善分类器分类性能的方法。

正约束(Positive constraint)和负约束(negative constraint)用来限制测试样本的标签赋值过程，而PN学习正是受正负约束所操控的。PN学习对分类器在测试样本上的分类结果进行评估，找到那些分类结果与约束条件相矛盾的样本，重新调整训练集，并进行重复迭代训练，直到某个条件满足，才停止分类器训练过程。在目标跟踪过程中，由于被跟踪目标的形状、姿态等容易发生变化，造成目标跟丢的情况时有发生，所以，在这种情况下，对被跟踪目标的在线学习和检测是个很好的策略。而PN学习正好可以在此处大显身手。

很多学习算法都假设测试样本是彼此独立的，然而，在计算机视觉的应用中，有些测试样本的标签却存在彼此依赖的关系。标签之间存在的这种依赖关系，我们称之为结构性的。例如，在目标检测过程中，我们的任务是对图片中目标可能存在的所有区域赋予标签，即：该区域属于前景或者背景，而这里的标签仅能是前景或背景两者之一。再比如，在利用视频序列进行目标跟踪过程中，紧邻被跟踪目标运动轨迹线的区域，可以认为是前景标签，而远离轨迹线的区域，可以认为是背景标签。而前面提到的正约束则表示所有可能的标签为正的模式，例如，此处的紧邻轨迹线的区域；负约束表示所有可能的标签为负的模式。

通过以上的分析，不难发现，PN学习可以定义为一下的过程：

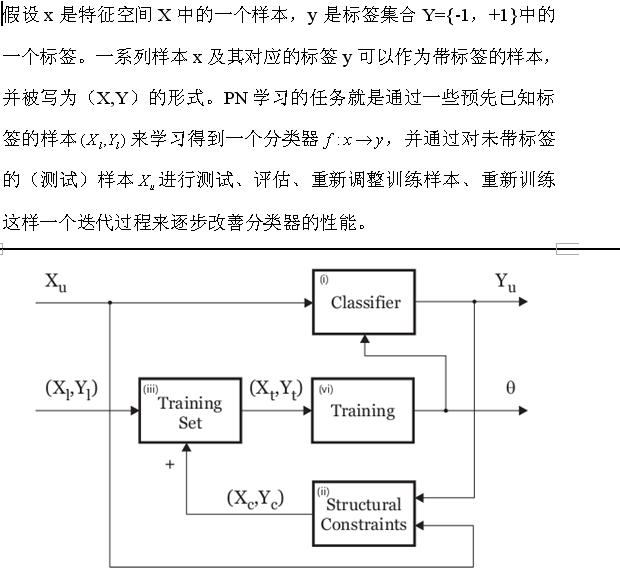
（1）准备一个数量较少的训练样本集合和一个数量很大的测试样本集合。

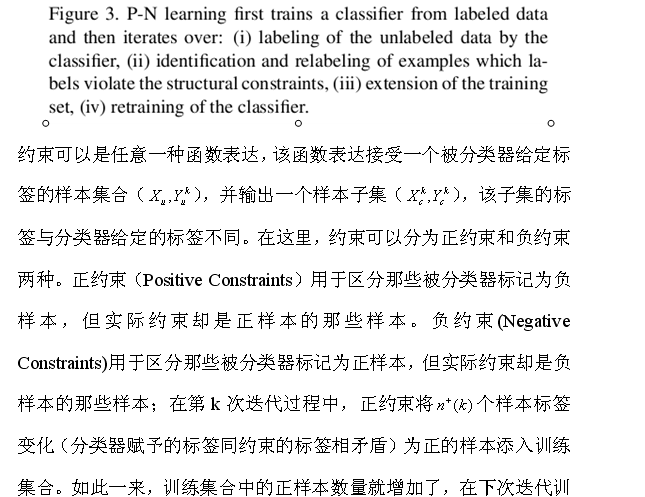
（2）利用训练样本训练一个初始分类器。同时，利用训练样本对(先验)约束条件进行相应的调整。

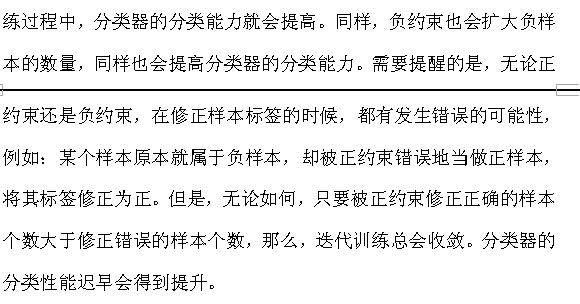
（3）利用分类器对测试样本赋予标签，并找出分类器赋予的标签同约束条件相矛盾的那些样本；

（4）将上述相矛盾的样本重新赋予标签，将其加入训练样本，重新训练分类器；

反复迭代上述过程，直到满足某个约束条件。







## [再谈PN学习](http://blog.csdn.net/carson2005/article/details/7647519)

       之前翻译过一篇PN学习的文章：<http://blog.csdn.net/carson2005/article/details/7483027>，但该文章的内容还是略显生涩，不太容易理解。尤其是在TLD跟踪算法中，PN学习又是一个很重要的模块。如果不能很好理解该部分，是很难完全掌握TLD算法精髓的。所以，这里我在上次翻译的基础上，结合TLD算法中的PN学习的具体应用，再次讲述PN学习的原理。

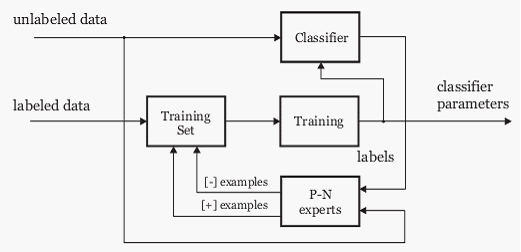
PN学习即PN learning, P指代Positive Constraint,也称之为P-expert或者growing event,N指代Negative Constraint,也称之为N-expert或者pruning event。

P-expert的作用是发现目标的新的外观（形变），并以此来增加正样本的数量，从而使得检测模块更具鲁棒性；

N-expert的作用是生成负的训练样本。N-expert的前提假设是，（被跟踪的）前景目标仅可能出现在视频帧中的一个位置，因此，如果前景目标的位置是确定的，那么其周围必然是负样例。

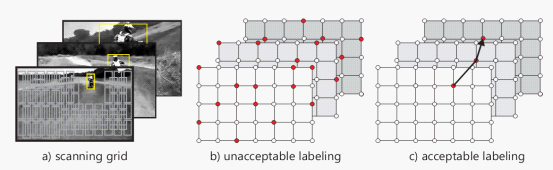
       TLD模块中的PN学习作用是通过对视频序列的在线处理来逐步改善检测模块(TLD中的Detection)的性能。对视频中的每一帧而言，我们希望评估检测模块在当前帧中的误检，并以此来更新目标模型，从而使得在以后的视频帧处理过程中避免类似的错误再次发生。PN学习的关键在于两种类型的“专家(experts)”：P-experts检查那些被检测模块错误分类为正样本（前景目标）的数据；N-experts检查哪些被检测模块错误分类为负样本(背景)的数据；需要提醒的是，无论P-experts还是N-experts都会产生一定的偏差。那么，如果用这些存在偏差的数据来更新检测模块（目标模型），是否会造成检测模型的性能恶化呢？作者经过研究发现，尽管存在误差，在一定条件下，误差是允许的，并且检测模块的性能会因此得到改善。

        PN学习包含四个部分：（1）一个待学习的分类器；（2）训练样本集--一些已知类别标签的样本；（3）监督学习--一种从训练样本集中训练分类器的方法；（4）P-N experts--在学习过程中用于产生正（训练）样本和负（训练）样本的表达函数；这四个部分之间的关系如下图所示：



        首先根据一些已有类别标记的样本，借助监督学习方法来训练，从而得到一个初始分类器。之后，通过迭代学习，利用上一次迭代得到的分类器对所有的未赋予标签的样本数据进行分类，而P-N experts则找出那些错误分类的样本，并依此来对训练样本集做出修正，使得下一次迭代训练之后得到的分类器的性能有所改善。P-experts将那些被分类器标记为负样本，但根据结构性约束条件应该为正样本的那些样本赋予“正”的标签，并添加到训练样本集中；而N-experts则将那些被分类器标记为正样本，但根据结构性约束条件应该为负样本的那些样本赋予“负”的标签，并添加到训练样本集当中；这也就意味着，P-experts增加了分类器的鲁棒性，而N-experts则增加了分类器的判别能力。

       下面我们就举例来说明PN学习的运行机制：假设存在三个连续的视频帧如下所示，每个视频帧之上都有若干个扫描窗口如(a)所示；



每一个扫描窗口就表示一个图像片(image patch)，图像片的类别标签用(b)(c)中的彩色圆点来表示。检测模块对每个图像片的类别赋值过程是彼此独立的，因此，N个扫描窗口就存在个类别标签的组合。而(b)则显示了其中一种可能的类别标签形式，这种类别标签标明，待检测目标在一个视频帧中可能同时出现在好几个区域，并且，待检测目标在相邻视频帧之间的运动没有连续性（例如(b)中最前面的图像中右上角的红色圆点在后面的两个图像中均没有出现），显然，这种类别标签形式是错误的。相反，(c)所示的类别标签形式则显示，每个视频帧中，目标只可能出现在一个区域，并且，相邻视频帧之间检测到的目标区域是连续了，构成了一个目标的运动轨迹。这种性质，我们称之为“结构性”的。PN学习的关键就是找到这种结构性的数据，从而来判别检测模块所产生的错误标签；

      刚才的例子表明：P-experts寻找视频序列中的时域上的结构性特征，并且假设目标是沿着轨迹线移动的，即，相邻帧之间的移动很小，且存在一定的相关性。P-experts记录目标在上一帧中的位置，并根据帧与帧之间的跟踪算法（这里采用的是LK光流法）来预测目标在当前帧中的位置。如果检测模块将跟踪算法预测到的目标在当前帧中的位置标记为负标签，那么P-experts就产生一个正的训练样本；N-experts寻找视频序列中的空间域上的结构性特征，并且假设目标在一个视频帧中只可能出现在一个位置。N-experts对检测模块在当前帧中的所有输出结果以及跟踪模块的输出结果进行分析，并找到具有最大可能性的那个区域。当前帧中所有目标可能出现的区域当中，如果某个区域同最大可能性区域之间没有重叠，就将其认定为负样本。另外，具有最大可能性的那个区域，被用于重新初始化跟踪模块；

        下面，再给出一个例子来说明情况：



        上图所示为三个连续的视频帧，PN学习需要处理的是t时刻那个黄色框所在区域的那个小车。跟踪模块在相邻帧之间给出小车的位置，从前面的分析我们知道，跟踪模块给出的区域可以被P-experts用来产生正的训练样本；但由于遮挡，在t+2时刻，P-experts产生了错误的正（训练）样本；与此同时，N-experts则找出目标最可能出现的位置（用红色的星号来标记），并且将所有其他的区域标记为负的训练样本；这里，N-experts在t+2时刻很好的修正了P-experts的错误；

# [TLD跟踪算法优化（一）并行化](http://blog.csdn.net/sunboyiris/article/details/34441213)

       才学疏浅，只言片语，只求志同道的朋友一起交流研究。

       并行化不算是算法的改进，只是追求运行的实时性。

       简要列举一个例子：

       TLD算法的C++版本源码里：

LKTracker::trackf2f(const Mat& img1, const Mat& img2,vector<Point2f> &points1, vector<cv::Point2f> &points2){

**[cpp]** [view plaincopy](http://blog.csdn.net/sunboyiris/article/details/34441213)[在CODE上查看代码片](https://code.csdn.net/snippets/405741)

1. **bool** LKTracker::trackf2f(**const** Mat& img1, **const** Mat& img2,vector<Point2f> &points1, vector<cv::Point2f> &points2){
2. //TODO!:implement c function cvCalcOpticalFlowPyrLK() or Faster tracking function
3. //Forward-Backward tracking
4. #pragma omp parallel sections //声明该并行区域分为若干个section,section之间的运行顺序为并行的关系
5. {
6. #pragma omp section //第一个section,由某个线程单独完成
7. //前向轨迹跟踪
8. calcOpticalFlowPyrLK( img1,img2, points1, points2, status,similarity, window\_size, level, term\_criteria, lambda, 0);

11. #pragma omp section //第二个section,由某个线程单独完成
12. //后向轨迹跟踪
13. calcOpticalFlowPyrLK( img2,img1, points2, pointsFB, FB\_status,FB\_error, window\_size, level, term\_criteria, lambda, 0);
14. }
15. //前向轨迹跟踪
16. // calcOpticalFlowPyrLK( img1,img2, points1, points2, status,similarity, window\_size, level, term\_criteria, lambda, 0);
17. //后向轨迹跟踪
18. //calcOpticalFlowPyrLK( img2,img1, points2, pointsFB, FB\_status,FB\_error, window\_size, level, term\_criteria, lambda, 0);
20. //Compute the real FB-error
21. /\*
22. 原理很简单：从t时刻的图像的A点，跟踪到t+1时刻的图像B点；然后倒回来
23. 从t+1时刻的图像的B点往回跟踪，假如跟踪到t时刻的图像的C点，这样就产
24. 生了前向和后向两个轨迹，比较t时刻中A点和C点的距离，如果距离小于某个
25. 阈值，那么就认为前向跟踪是正确的；这个距离就是FB\_error
26. \*/
27. //计算前向与后向轨迹的误差。
28. #pragma  omp parallel for
29. **for**( **int** i= 0; i<points1.size(); ++i ){
30. FB\_error[i] = norm(pointsFB[i]-points1[i]); //norm求矩阵或向量的
31. //范数,或绝对值
32. }
33. //Filter out points with FB\_error[i] > median(FB\_error) && points with sim\_error[i] > median(sim\_error)
34. normCrossCorrelation(img1,img2,points1,points2);
35. **return** filterPts(points1,points2);
36. }

 修改后代码运行速度提高了不少。

 不过并行化处理，必须考虑到一些问题

1.数据的互斥问题

2.线程的分配问题

3.Release版本应用程序对于for循环可以自动优化，不用对for做多线程设定，主要还是放在模块化的数据处理并行化上。

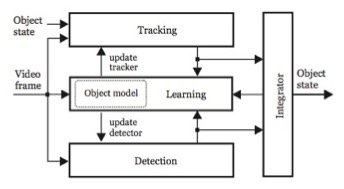
# [TLD跟踪算法学习及其在摄像机控制中的应用](http://blog.csdn.net/qianxin_dh/article/details/40221883)

 最近一段时间在学习TLD跟踪算法，通过对代码的解读以及实际运行，感觉到这个算法的跟踪效果那是十分犀利啊！真想拿它来做点小东西，满足下自己的好奇心，嘿嘿～趁着实验室有装备，我就搭建了一个小小系统，利MFC做了个界面，将TLD算法作为主要核心，实现了摄像机随着人物的移动而跟着相应摆动，当然，本人水平有限，目前效果算是一般，接下来想把这个系统接着完善下去，同时程序因为涉及到相机控制等，代码量比较大，这里就不贴出来了～～ 好啦，下面也将这段时间的学习记录下。

## 一.TLD概述

TLD算法将长期目标跟踪任务分解为三个模块：跟踪模块，检测模块，学习模块。

其中，跟踪模块主要负责对相邻帧间的目标进行跟踪；检测模块会找出一系列目标可能会出现的位置，并在必要时对跟踪模块进行校正(如跟踪失败时),学习模块则是对检测器的性能进行评估并更新，TLD框架如图一所示。



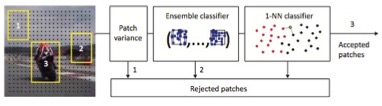
图一

跟踪模块：

TLD中采用了金字塔光流法，并结合跟踪错误检测算法(FB error及匹配相似度)实现对目标的跟踪任务。

检测模块（个人认为这里有很大的改进空间，比如降低候选目标窗口数量，对分类器进行精简等，最终在跟踪的准确性和实时性上找到一个balance）：

TLD的检测分类器(图二所示)包括三部分：方差分类器，集合分类器，最近邻分类器。当前帧中的每一个扫描窗口依次通过上面三个分类器，全部通过才被认为含有前景目标。



图二

方差分类器：利用积分图计算每个扫描窗口的方差，方差大于某一阈值(目标图像片方差的50%)的，进入集合分类器模块。

集合分类器：集合分类器共有10棵树(基本分类器)，每棵树有13个判断节点，每个判断节点经过比较得到一个二进制位0或者1，这样每棵树就对应得到一个13位的二进制码x。每个二进制码x对应于一个后验概率，那么整个集合分类器就有10个后验概率，将10个后验概率进行平均，如果大于阈值(经验值0.65)，该图像片含有前景目标

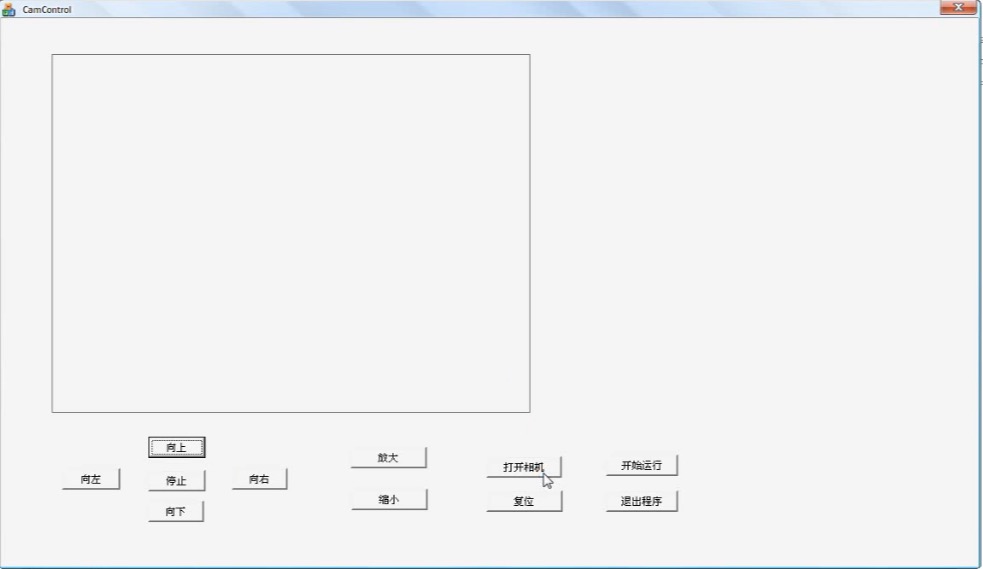
最近邻分类器：计算图像片(归一化为15\*15)到在线模型的相关相似度与保守相似度，若相关相似度大于某阈值(0.6)，则认为含有前景目标。

综合模块综合跟踪器跟踪到的单个目标和检测器可能检测到的多个目标，然后输出保守相似度最大的一个目标。

学习模块：

TLD学习模块主要生成集合分类器和最近邻分类器所需的样本，对分类器进行训练。

## 二.TLD在跟踪系统中的应用说明



        首先，大家不要喷我怎么把界面设计的如此难堪(初步设计一下...)。这里我也只是简要介绍下这个界面的功能(有兴趣的朋友可以和我多多交流，大家共同进步)，界面的上方是用于显示跟踪画面，也就是picture控件，界面的下方有一些按钮，分别是对摄像机的操作(如上下左右，放大缩小...)，控制换面显示的开关以及跟踪算法的运行，空白处是我当时预留的位置，想加入的一些别的功能，有机会再补充吧。这些摄像机控制按钮只是一开始为了调整摄像机位置设置的，等程序开始运行后，便不再需要操作，摄像机会根据目标的位置自动摆动到相应位置，而且个人感觉速度还马马虎虎吧。

# [计算机视觉 之 在iOS上测试跟踪算法Visual Object Tracking Algorithm](http://blog.csdn.net/songrotek/article/details/47422057)

### 前言

在计算机视觉CV领域，视觉跟踪是其中一个重要的子问题。从我的角度看，视觉跟踪就是用在机器人上，用在移动设备上，因此，何不把一些跟踪算法放在iPhone上看看实际的跟踪效果。这才是最真实的对比，使用一些视频毕竟不实际，而且关键是不能很好的对比实时性。对于移动设备而已，实时性是最重要的吧。之所以有跟踪问题的出现，也是因为绝大多数的物体检测Object Detection算法实在是太慢了，根本无法用在视频当中。因此才有了Object Tracking的问题。最终的目标也就是Real Time Visual Object Tracking。现在的很热的四轴无人机的研究热点不就是自动跟踪物体嘛。但是这完全取决于跟踪算法的水平。难度确实是大。

这里我使用iPhone来做测试。iPhone的性能在移动设备上也不错。因此看他的算法效果还是不错的，关键是方便，想跟踪个什么东西直接在手机屏幕画个框就ok了。

其次现在的跟踪问题都变成了在视频上初始画个框，然后要求框一直跟着初始框内的物体。其实这种简化的问题描述是有问题的，但不妨碍CV界如火如荼的研究。

### 使用的跟踪算法

直接放代码了：   
<https://github.com/songrotek/Tracking-on-iOS>

里面包含了CMT，TLD，Color Tracking，CT，Camshift，Struck跟踪算法。具体算法我这里没办法细说，具体细节我也是在研究，只不过是把其C++源代码合到xcode中实现了。   
这中间还是耗费了很多的精力，分享出来希望对也在研究CV的童鞋有帮助！

下面是CMT算法的效果，其实挺赞的！实时性很不错！   
   
   
