# [TLD之扯淡篇](http://blog.csdn.net/ttransposition/article/details/43195097)

*Reference:*

**[1] Tracking-Learning-Detection,PAMI 2012**

**[2] Forward-Backward Error: Automatic Detection of Tracking Failures,ICPR 2010**

**Code**: [Matlab](https://github.com/zk00006/OpenTLD)，[C++](https://github.com/zk00006/OpenTLD)，[我的完整注释（C++）](http://download.csdn.net/detail/soidnhp/8401005)

**作者主页**：[资料完备](http://www.cvchina.info/tag/tracking-learning-detection/)、[官方](http://personal.ee.surrey.ac.uk/Personal/Z.Kalal/)、[TLD 1.0&2.0](http://www.tldvision.com/)

**前人的部分博客：**

[**TLD算法流程说明--episode**](http://blog.sina.com.cn/s/blog_70a384770101m8eg.html)

[**Tracking-Learning-Detection TLD解析**](http://blog.csdn.net/wood_water/article/details/9017681)

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



电影《极品飞车》

    经常看大片的人，对于这样的场景，一定不会陌生：猪脚（车技一流）驱车逃逸，遭N多警车围追堵截，与此同时，天上还会有直升飞机，场面异常震撼。好了，问题来了，大家有没有想过，为什么派出警车的同时还要派出直升飞机呢？我觉得吧，作为电影造势是必须的，还有就是空中打击具有无可比拟的优势。不过，今天我只从目标跟踪角度看待这个问题，从而引出今天的猪脚，**TLD目标跟踪算法**。

     假设我们的目的是要地面抓捕，那首先得保证能跟踪到目标，待时机成熟，再实施抓捕，从跟踪的角度来讲，直升飞机是有助于跟踪目标的。待小生细细道来：

    Case 1: 对于车辆少，道路分叉少的情况下，警车只要用肉眼便可跟上目标，直升飞机显得多余，当然，这是比较理想的情况，不过身为猪脚，肯定没这么笨。

    Case 2:，于是他钻到了繁华的商业区，车多，岔路多，导致警车的行动不便，视野也容易被遮挡。在一个高楼林立的三岔路口，一辆飞驰而过的救护车，成功地掩护了我们的猪脚，于是猪脚就跟丢了……，但是呢，在直升机眼皮底下，想逃走就没那么容易了。于是，警车在直升机的引导下，重新找到了我们那脚底抹了香油的猪脚。抽象一下，车的可视范围是一条线，而直升飞机的可视范围是面。

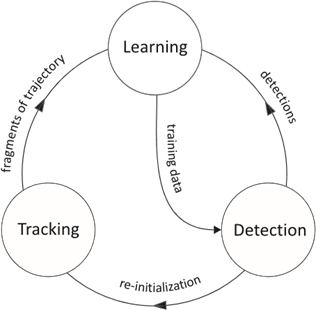
    Case 3: 不过，身为猪脚，必须高智商，为了躲避直升飞机，它中途挟持了一个听话的配角，于是他俩愉快地换车了。由于其迅雷不及掩耳之速，加上直升飞机距离地面太远，猪脚又很小，因而视物模糊，结果，直升飞机并没有察觉。不幸的是，警车里面的人可都看在眼里呀（假设哈）……于是警车里的人及时向直升飞机里的人反馈，于是直升飞机又重新锁定了目标。

## TLD概述

    目标跟踪的应用场合很多，很多也挺有意思，下面这张图是我从[这里](http://blog.csdn.net/wood_water/article/details/9017681)盗来的……



    TLD是一个比较有名的目标跟踪算法，作者甚至用这个算法开了一个[公司](http://tldvision.com/)。TLD的思路和上面漏洞百出的例子（不过，我从中得到了一些提高速度的启示）很相似，它有三大猪脚，Tracker（跟踪，简称小T），Detector（检测器，小D），Leaner（简称小L）。小T就相当于警车，小D就相当于直升飞机，小L不太好打比方，总之，凡是要综合小D和小T的信息，就是小L要干的活，比如小T跟丢了之后，小D告诉他目标的位置。TLD算法成功的原因就在于它将检测器和跟踪器有机的整合在一起，从而实现了**长线跟踪**。大伙先看看一个[video demo](http://v.youku.com/v_show/id_XMjcxNTczOTMy.html?from=y1.2-1-176.3.1-1.1-1-1-0),从视频中可以看得出来TLD是一种**单目标跟踪算法**，操作很简单，只需要框定要跟踪的目标即可。



TLD框图

    下面介绍TLD的这三大主角，小T，小D，小L，ppt里面的介绍如下：

**Tracker** estimates the object motion under the assumption that the object is **visible** and its **motion is limited**.

**Detector** performs **full scanning of the image** to localize all appearances that have been **observed in the past**.

**Learning** **observes performance of both**, the tracker and the detector, **identifies errors** of the detector and generates training examples to avoid these errors in the future.

    补充说明一下：

    小T是依据上一帧和上一帧中目标所处的位置，来预测当前帧目标出现的位置，具体是使用光流法。所以，小T和警车很像，眼睛得一直盯着车。

    首先注意到小D是使用**多尺度sliding windows(滑动窗口)检测方法**，所以检测可以看成一个**广义的分类问题**，于是就需要正负样本来训练分类器。由于目标会不断地变化，比如，跟踪人脸的时候，闭上眼睛，张开嘴巴等变化，这就需要分类器还不仅要认得原始目标而且还得具有一双火眼金睛，能认出目标的七十二般变化，为了修炼这样一双眼睛，就需要把这些变化后的样本加入训练集重新训练分类器，从而让**分类器随着目标的变化而不断更新**。所以，小D很像盘旋在天空的直升飞机，他只需要瞟一眼就能找到目标，所以他侧重找目标，而不是跟踪。

    小L的作用是**综合**小D和小T的跟踪结果得到目标所处的区域，同时它还要让**小D和小T这两兄弟相互学习**。

    既然是兄弟，我喜欢分个长幼，哪个靠谱哪个为兄。显然，相对于跟踪的结果而已，检测器的结果一般更靠谱，为什么呢？首先检测器的搜索范围是全局的，找出来的目标是全局最优的，而检测器是局部的，其次，检测器是经过严格训练的，其判断依据是所有历史数据，而跟踪器只不过依据上下两帧的孤立跟踪点，而目标的运动是变幻莫测的，所以**小D是兄，小T是弟**。

    当小D的表现比小T好的时候，小T就得**接受**小D的结果，即小T要放弃它自己的跟踪结果，接受小D的结果，于是下一帧小T就从小D所认为的位置开始跟踪，对应TLD框图中的re-initialization。

   但是，当小T的结果比小D好的时候，小T得尊敬兄长，所谓长兄为父，所以目标最终的位置，并不一定是小T说了算，它还得**参考**小D结果。小D表现不好分为三种情况，后面细说。当然，**小D还是要跟小T学习**，对应TLD框图中的trianing data。

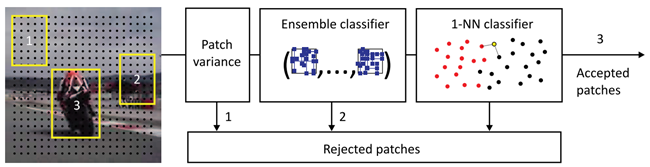
    这里只是泛泛地说明了两种情况，实际上的情况非常多，后面再详细讨论小L。

本文分析的是C++版本，首先记住几个类TLD、 LKTracker、FerNNClassifier，大致对应小L，小T，小D。

# [TLD之检测篇](http://blog.csdn.net/ttransposition/article/details/43195765)

    扫描方式前面已经说过，具体参数【5.3】：scales step =1.2, horizontal step =10 percent of width, vertical step =10 percent of height, minimal bounding box size = 20 pixels. This setting produces around 50k bounding boxes for a QVGA image (240×320), the exact number depends on the aspect ratio of the initial bounding box.

    TLD分类器的三道关卡

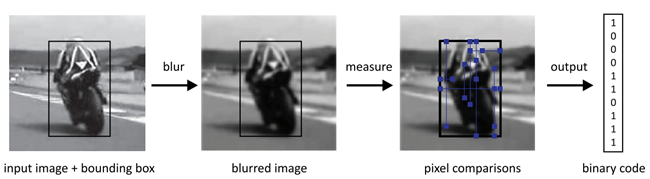


    如上图所示，TLD的检测算法共有三关：

## 第一关：方差

    使用简单的阈值判断，阈值设定为初始选择目标方差的50%（存储在TLD::var），见【5.3.1】50 percent of variance of the patch that was selected  for tracking。

## 第二关：随机森林分类器



    TLD特征

    首先介绍一下TLD所使用的特征，2bit BP，很简单，就是任意两个点的大小关系，取值只有0和1。结合后面的分类器，我更倾向于将特征定义为0/1组成的串/向量。具体来   说，首先随机产生13对坐标，然后比较对应坐标像素值的大小，得到13个0/1，最后依次组成13位二进制数，也就可以看出一个整数。为了去噪，预先对图像进行了高斯滤波。不过图TLD特征中，只有10位。

    TLD所使用的随机森林相当简单，一共有10棵树，每一棵树的分类特征就是上述的13位二进制串，与CART，ID3等决策树不同，它没有特征选择，它只是对特征分布进行了**直方图统计**，所以，每一个树可以看成一个贝叶斯分类器，而且直方图的区间数多到2^13。每一棵树分类的方法特么简单，正样本的概率为特征对应的区间**正负样本数之比**。最后对所有树的分类**结果求和**即为随机森林的最终分类结果，即正样本的概率。原文【5.3.2】说结果的平均值超过0.5就当做正样本，不过，实际代码中是超过0.6才算正样本，而结果超过0.5的负样本才作为随机森林重新训练的负样本，即hard negative。

    下面直接看源码实现

FerNNClassifier::prepare实现确定**特征的比对位置**，它在TLD::init函数中被调用,主要是这几个变量**features[s][i]、thrN、 posteriors、pCounter、nCounter，**命名非常直白，含义就不多说了。

**[cpp]** [view plaincopy](http://blog.csdn.net/ttransposition/article/details/43195765)[在CODE上查看代码片](https://code.csdn.net/snippets/590447)

1. **void** FerNNClassifier::prepare(**const** vector<Size>& scales){
2. acum = 0;
3. // 1. Initialize test locations for features
4. //   随机产生需要坐标对（x1f，y1f，x2f，y2f，注意范围[0,1)），
5. //   即确定由每一个特征是由哪些点对进行而得到，这些位置一旦确定就不会改变，
6. //   由于我们要进行多尺度检测，所以同一个点在不同尺度scales，实际对应的坐标要乘以对应尺度的width和height。
7. **int** totalFeatures = nstructs\*structSize;//nstructs 10 structSize 13
8. features = vector<vector<Feature> >(scales.size(),vector<Feature> (totalFeatures));
9. RNG& rng = theRNG();
10. **float** x1f,x2f,y1f,y2f;
11. **int** x1, x2, y1, y2;
12. **for** (**int** i=0;i<totalFeatures;i++){
13. x1f = (**float**)rng; //产生[0,1)直接的浮点数
14. y1f = (**float**)rng;
15. x2f = (**float**)rng;
16. y2f = (**float**)rng;
17. **for** (**int** s=0;s<scales.size();s++){
18. x1 = x1f \* scales[s].width;
19. y1 = y1f \* scales[s].height;
20. x2 = x2f \* scales[s].width;
21. y2 = y2f \* scales[s].height;
22. features[s][i] = Feature(x1, y1, x2, y2);
23. }
24. }
25. // 2. Thresholds，负样本的阈值
26. thrN = 0.5\*nstructs;
27. // 3. Initialize Posteriors，为统计直方图分配空间
28. **for** (**int** i = 0; i<nstructs; i++) {
29. posteriors.push\_back(vector<**float**>(pow(2.0,structSize), 0));
30. pCounter.push\_back(vector<**int**>(pow(2.0,structSize), 0));
31. nCounter.push\_back(vector<**int**>(pow(2.0,structSize), 0));
32. }
33. }

    确定了由哪些点对得到特征后，就可以**获取给定图像块（patch）的特征**了，scale\_idx是图像块的尺度索引，fern[t]就是所提取的第t个特征，前面说过，特征是一个13位二进制数，也就是一个整数。

**[cpp]** [view plaincopy](http://blog.csdn.net/ttransposition/article/details/43195765)[在CODE上查看代码片](https://code.csdn.net/snippets/590447)

1. **void** FerNNClassifier::getFeatures(**const** cv::Mat&image,constint& scale\_idx, vector<**int**>& fern){
2. **int** leaf;
3. **for** (**int** t=0;t<nstructs;t++){
4. leaf=0;
5. **for** (**int** f=0; f<structSize; f++){
6. //依次得到每一位
7. leaf = (leaf << 1) + features[scale\_idx][t\*nstructs+f](image);
8. }
9. fern[t]=leaf;
10. }
11. }

     能够提取特征了，那么接下来就可以**训练随机森林分类器**，这部分稍微分的有点细，首先介绍训练函数FerNNClassifier::trainF(ferns,resample)，ferns是训练集，即正/负样本的特征，不过呢，类别标号ferns[i].second==1即为正样本，resample是**bootstrap的次数，其实**函数实现时只有一轮，**bootstrap**是用容易分错的正样本和负样本更新分类器，measure\_forest就是该分类器的分类函数，update是更新函数。

**[cpp]** [view plaincopy](http://blog.csdn.net/ttransposition/article/details/43195765)[在CODE上查看代码片](https://code.csdn.net/snippets/590447)

1. **void** FerNNClassifier::trainF(**const** vector<std::pair<vector<**int**>,**int**> >&ferns,intresample){
2. thrP = thr\_fern\*nstructs; //0.6\*10
3. **for** (**int** i = 0; i <ferns.size(); i++){
4. **if**(ferns[i].second==1){//正样本
5. **if**(measure\_forest(ferns[i].first)<=thrP)
6. update(ferns[i].first,1,1);
7. }**else**{//负样本
8. **if** (measure\_forest(ferns[i].first) >= thrN)
9. update(ferns[i].first,0,1);
10. }
11. }
12. }

FerNNClassifier::**measure\_forest**就是前面提到的，将10棵树的概率求和

**[cpp]** [view plaincopy](http://blog.csdn.net/ttransposition/article/details/43195765)[在CODE上查看代码片](https://code.csdn.net/snippets/590447)

1. **float** FerNNClassifier::measure\_forest(vector<**int**>fern) {
2. **float** votes = 0;
3. **for** (**int** i = 0; i < nstructs; i++) {
4. votes += posteriors[i][fern[i]];
5. }
6. **return** votes;
7. }

**update**更新正负样本的直方图分布，注意：posteriors只算了正样本的概率

**[cpp]** [view plaincopy](http://blog.csdn.net/ttransposition/article/details/43195765)[在CODE上查看代码片](https://code.csdn.net/snippets/590447)

1. **void** FerNNClassifier::update(**const** vector<**int**>& fern,intC, **int** N) {
2. **int** idx;
3. **for** (**int** i = 0; i < nstructs; i++) {//10
4. idx = fern[i];//13位的特征
5. //C=1，正样本，C=0，负样本
6. (C==1) ? pCounter[i][idx] +=N : nCounter[i][idx] +=N;
7. **if** (pCounter[i][idx]==0) {//既然是正概率，如果正样本的数目为0，正样本的概率自然也为0
8. posteriors[i][idx] = 0;
9. } **else** {
10. posteriors[i][idx] = ((**float**)(pCounter[i][idx]))/(pCounter[i][idx] + nCounter[i][idx]);
11. }
12. }
13. }

## 第三关：最近邻分类器

**最近邻分类器**顾名思义咯，与随机森林一样，其中也涉及到特征、训练函数、分类函数。

**特征**其实就是将**图像块**大小归一化（都变成patch\_size×patch\_size），零均值化【5.1】

**[cpp]** [view plaincopy](http://blog.csdn.net/ttransposition/article/details/43195765)[在CODE上查看代码片](https://code.csdn.net/snippets/590447)

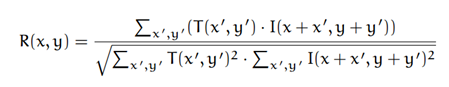
1. **void** TLD::getPattern(**const** Mat& img, Mat& pattern,Scalar&mean,Scalar&stdev){
2. resize(img,pattern,Size(patch\_size,patch\_size));
3. meanStdDev(pattern,mean,stdev);
4. pattern.convertTo(pattern,CV\_32F);
5. pattern = pattern-mean.val[0];
6. }

    训练函数trainNN，我觉得最近邻分类器其实没有所谓的训练，因为只需要将容易分错的正/负样本**加入**正负样本集就可以了。其中，pEx是正样本集，nEx是负样本集。

**[cpp]** [view plaincopy](http://blog.csdn.net/ttransposition/article/details/43195765)[在CODE上查看代码片](https://code.csdn.net/snippets/590447)

1. **void** FerNNClassifier::trainNN(**const** vector<cv::Mat>& nn\_examples){
2. **float** conf,dummy;
3. vector<**int**> y(nn\_examples.size(),0);
4. y[0]=1;//只有第一个是正样本，并不是原始的目标区域，而是best\_box
5. vector<**int**> isin;
6. **for** (**int** i=0;i<nn\_examples.size();i++){//  For each example
7. NNConf(nn\_examples[i],isin,conf,dummy);// Measure Relative similarity
8. **if** (y[i]==1 && conf<=thr\_nn){
9. **if** (isin[1]<0){ //注意：如果pEx为空，NNConf直接返回 thr\_nn=0，isin都为-1，
10. pEx = vector<Mat>(1,nn\_examples[i]);
11. **continue**;
12. }
13. pEx.push\_back(nn\_examples[i]);//之前存在正样本，追加
14. }
15. **if**(y[i]==0 && conf>0.5)
16. nEx.push\_back(nn\_examples[i]);
17. }
18. acum++;
19. printf("%d. Trained NN examples: %d positive %d negative\n",acum,(**int**)pEx.size(),(**int**)nEx.size());
20. }

    分类函数NNConf，计算的就是待分类样本example和NN分类器中所有正负样本的距离，距离是酱紫计算的（见OpenCV refermanual）：



    好吧，这个我也是第一次见，不过这个和**相关系数**特别像：

http://images.cnitblog.com/blog/460184/201501/271548448164451.png

http://images.cnitblog.com/blog/460184/201501/271548452848995.png

   区别是没有减去均值，还记得我们前面提到图像块都进行了零均值化，因此**距离就是计算相关系数**……

    不过，还要进行一些处理才方便作为距离测度, 相关系数的取值范围是[-1,1]，加上1变成[0,2]，再**将范围缩小为[0,1]**

**http://images.cnitblog.com/blog/460184/201501/271548455343539.png**

    相似性包含两种，Relative similarity和Conservative similarity，具体见【5.2】，**不过这个版本采用了另一种计算方式，大家自己领会一下吧，我也说不上哪个好。**

**[cpp]** [view plaincopy](http://blog.csdn.net/ttransposition/article/details/43195765)[在CODE上查看代码片](https://code.csdn.net/snippets/590447)

1. **void** FerNNClassifier::NNConf(**const** Mat& example, vector<**int**>& isin,**float**&rsconf,**float**&csconf){
2. isin=vector<**int**>(3,-1);
3. **if** (pEx.empty()){ //if isempty(tld.pex) % IF positive examples in the model are not defined THEN everything is negative
4. rsconf = 0; //    conf1 = zeros(1,size(x,2));
5. csconf=0;
6. **return**;
7. }
8. **if** (nEx.empty()){ //if isempty(tld.nex) % IF negative examples in the model are not defined THEN everything is positive
9. rsconf = 1;   //    conf1 = ones(1,size(x,2));
10. csconf=1;
11. **return**;
12. }
13. Mat ncc(1,1,CV\_32F);
14. **float** nccP,csmaxP,maxP=0;
15. **bool** anyP=**false**;
16. **int** maxPidx,validatedPart = ceil(pEx.size()\*valid);//正样本的前 50%，用于计算Conservative similarit【5.2 5】
17. **float** nccN, maxN=0;
18. **bool** anyN=**false**;
19. **for** (**int** i=0;i<pEx.size();i++){
20. matchTemplate(pEx[i],example,ncc,CV\_TM\_CCORR\_NORMED);// measure NCC to positive examples
21. //相关系数的取值范围是[-1,1]，加上1变成[0,2]，再将范围缩小为[0,1]
22. nccP=(((**float**\*)ncc.data)[0]+1)\*0.5;
23. **if** (nccP>ncc\_thesame)//0.95
24. anyP=**true**;
25. **if**(nccP > maxP){
26. maxP=nccP;//Relative similarity
27. maxPidx = i;
28. **if**(i<validatedPart)
29. csmaxP=maxP;//Conservative similari
30. }
31. }
32. **for** (**int** i=0;i<nEx.size();i++){
33. matchTemplate(nEx[i],example,ncc,CV\_TM\_CCORR\_NORMED);//measure NCC to negative examples
34. nccN=(((**float**\*)ncc.data)[0]+1)\*0.5;
35. **if** (nccN>ncc\_thesame)
36. anyN=**true**;
37. **if**(nccN > maxN)
38. maxN=nccN;
39. }
40. //set isin
41. **if** (anyP) isin[0]=1;  //if he query patch is highly correlated with any positive patch in the model then it is considered to be one of them
42. isin[1]=maxPidx;      //get the index of the maximall correlated positive patch
43. **if** (anyN) isin[2]=1;  //if  the query patch is highly correlated with any negative patch in the model then it is considered to be one of them
44. //Measure Relative Similarity
45. **float** dN=1-maxN;
46. **float** dP=1-maxP;
47. rsconf = (**float**)dN/(dN+dP); //与原文【5.2】有出入，不过也是可以理解的
48. //Measure Conservative Similarity
49. dP = 1 - csmaxP;
50. csconf =(**float**)dN / (dN + dP);
51. }

## TLD::detect函数

   有了前面的铺垫，这段程序应该比较好懂了吧。

**[cpp]** [view plaincopy](http://blog.csdn.net/ttransposition/article/details/43195765)[在CODE上查看代码片](https://code.csdn.net/snippets/590447)

1. **void** TLD::detect(**const** cv::Mat&frame){
2. //cleaning
3. dbb.clear();
4. dconf.clear();
5. dt.bb.clear();//检测的结果，一个目标一个bounding box
6. **double** t = (**double**)getTickCount();
7. Mat img(frame.rows,frame.cols,CV\_8U);
8. integral(frame,iisum,iisqsum);//
9. GaussianBlur(frame,img,Size(9,9),1.5);//
10. **int** numtrees = classifier.getNumStructs();// nstructs： 10
11. **float** fern\_th = classifier.getFernTh();//thr\_fern：0.6
12. vector <**int**> ferns(10);
13. **float** conf;
14. **int** a=0;
15. Mat patch;
16. // 1. 方差->结果存在tmp ->随机森林-> dt.bb
17. **for** (**int** i=0;i<grid.size();i++){//FIXME: BottleNeck
18. **if** (getVar(grid[i],iisum,iisqsum)>=var){//第一关：方差
19. a++;
20. patch = img(grid[i]);
21. classifier.getFeatures(patch,grid[i].sidx,ferns);//sidx:scale index
22. conf = classifier.measure\_forest(ferns);//第二关：随机森林
23. tmp.conf[i]=conf; //只要能通过第一关就会保存到tmp
24. tmp.patt[i]=ferns;
25. **if** (conf>numtrees\*fern\_th){
26. dt.bb.push\_back(i); //第二关
27. }
28. }
29. **else**
30. tmp.conf[i]=0.0;//第一关都没过
31. }
32. **int** detections = dt.bb.size();
33. printf("%d Bounding boxes passed the variance filter\n",a);
34. printf("%d Initial detection from Fern Classifier\n",detections);
35. **if** (detections>100){//第二关附加赛：100名以后的回家去
36. nth\_element(dt.bb.begin(),dt.bb.begin()+100,dt.bb.end(),CComparator(tmp.conf));
37. dt.bb.resize(100);
38. detections=100;
39. }
40. **if** (detections==0){
41. detected=**false**;
42. **return**;//啥都没看到……
43. }
44. printf("Fern detector made %d detections ",detections);
45. t=(**double**)getTickCount()-t;
46. printf("in %gms\n", t\*1000/getTickFrequency());
47. //  Initialize detection structure
48. dt.patt = vector<vector<**int**> >(detections,vector<**int**>(10,0));       //  Corresponding codes of the Ensemble Classifier
49. dt.conf1 = vector<**float**>(detections);                               //  Relative Similarity (for final nearest neighbour classifier)
50. dt.conf2 =vector<**float**>(detections);                                //  Conservative Similarity (for integration with tracker)
51. dt.isin = vector<vector<**int**> >(detections,vector<**int**>(3,-1));       //  Detected (isin=1) or rejected (isin=0) by nearest neighbour classifier
52. dt.patch = vector<Mat>(detections,Mat(patch\_size,patch\_size,CV\_32F));//  Corresponding patches,patch\_size: 15
53. **int** idx;
54. Scalar mean, stdev;
55. **float** nn\_th = classifier.getNNTh();//thr\_nn:0.65
56. //3. 第三关：最近邻分类器，用Relative Similarity分类，但是却用 Conservative Similarity作为分数->dconf
57. **for** (**int** i=0;i<detections;i++){                                        //  for every remaining detection
58. idx=dt.bb[i];                                                      //  Get the detected bounding box index
59. patch = frame(grid[idx]);
60. getPattern(patch,dt.patch[i],mean,stdev);                //  Get pattern within bounding box
61. classifier.NNConf(dt.patch[i],dt.isin[i],dt.conf1[i],dt.conf2[i]); //  Evaluate nearest neighbour classifier
62. dt.patt[i]=tmp.patt[idx];//ferns
63. **if** (dt.conf1[i]>nn\_th){                                              //  idx = dt.conf1 > tld.model.thr\_nn; % get all indexes that made it through the nearest neighbour
64. dbb.push\_back(grid[idx]);                                        //  BB    = dt.bb(:,idx); % bounding boxes
65. dconf.push\_back(dt.conf2[i]);                                     //  Conf  = dt.conf2(:,idx); % conservative confidences
66. }
67. }                                                                        //  end
68. **if** (dbb.size()>0){
69. printf("Found %d NN matches\n",(**int**)dbb.size());
70. detected=**true**;
71. }
72. **else**{
73. printf("No NN matches found.\n");
74. detected=**false**;
75. }

# [TLD之跟踪篇](http://blog.csdn.net/ttransposition/article/details/43195851)

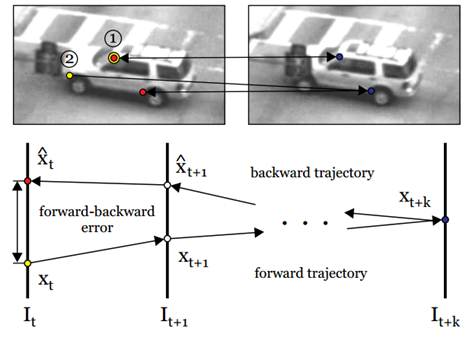
    目标跟踪的一般思想是**跟踪目标中关键点**。TLD也是**跟踪点**（但不是跟踪SIFT之类的关键点）。点跟踪采用的是**光流法**，具体来说是Pyramidal Lucas-Kanade tracker,这个以后机会再介绍，推荐阅读《Learning OpenCV》第10章的Lucas-Kanade Method部分，这里只介绍OpenCV的实现函数，跳过原理和实现细节。

    首先看**跟踪点**的函数，calcOpticalFlowPyrLK，它的作用是**找到上一帧中的跟踪点在当前帧的位置。**调用形式如下：

calcOpticalFlowPyrLK( img\_last,img\_curr, points\_last , points\_curr, status, errs)

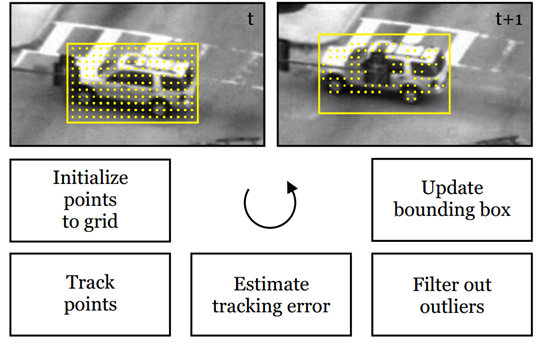
    参数意思应该很直白了吧，补充一下status为1，表示对应点找到了，为0就是没找到，errs自然是误差。注意：可以是单个点，也可以是点集，如果是点集，那么对应的status和errs就都是vector啦。

    下面说说怎么**跟踪目标**，TLD采用的是基于作者自己提出的Median-Flow tracker，此外增加了跟踪失败检测。



通过Forward-Backward error来筛选要跟踪的点

    前面提到TLD跟踪的不是关键点，它跟踪的是更简单的点：能**稳定存在的点**，那哪些点是稳定的呢？Median-Flow tracker的基本思想是，看反向跟踪后的残差，用所有点的残差中值作为稳定点的筛选条件。如上图中的黄色点就因为残差太大，被pass掉了，既然稳定点是可以筛选出来的，那么就不必煞费苦心的寻找那些关键点，可以直接**将所有的点都作为初始跟踪点，**好吧所，有的点毕竟还是太多了，于是作者是选取**网格交叉点**作为**初始跟踪点（见下图框框中黄色的点点）。**



Median Flow tracker 的流程图

    下面正式介绍作者的跟踪函数TLD::track，调用形式如下：

track(img1,img2,points1,points2)

    img1是上一帧的图像，img2是当前帧的图像，points1，points2都是这个函数的输出函数，points1是将上一次跟踪到的目标区域lastbox划分成网格后，所得到的网格交点，即上图左边的黄色点，而points2是points1中能稳定出现在当前帧出的点，即右图中的点。

    下面结合上面的流程图，并补充TLD需要增加的环节，来介绍track。

## ****TLD::track函数****

**[cpp]** [view plaincopy](http://blog.csdn.net/ttransposition/article/details/43195851)[在CODE上查看代码片](https://code.csdn.net/snippets/590455)

1. **void** TLD::track(**const** Mat& img1, **const** Mat& img2,vector<Point2f>& points1,vector<Point2f>& points2){
2. //【5.4】
3. // 1.Generate points
4. bbPoints(points1,lastbox);
5. **if** (points1.size()<1){//问题：何时会出现这种情况？？
6. printf("BB= %d %d %d %d, Points not generated\n",lastbox.x,lastbox.y,lastbox.width,lastbox.height);
7. tvalid=**false**;
8. tracked=**false**;
9. **return**;
10. }
11. vector<Point2f> points = points1;
12. //Frame-to-frame tracking with forward-backward error cheking
13. // 2. 推断上一帧的points，在当前帧的位置，points->points2
14. // 注意:只有通过筛选的point对 还保留在points，points2
15. tracked = tracker.trackf2f(img1,img2,points,points2);
16. **if** (tracked){//只要有一个点跟到了，就算跟到了……，是不是应该严格一点呢？？
17. // 3. Bounding box prediction
18. bbPredict(points,points2,lastbox,tbb);//此时，lastbox，还是依据上一帧预测的目标在当前帧的位置
19. <span style="white-space:pre">    </span>  // 4. Failure detection,检测 getFB()>10 || 完全出轨
20. **if** (tracker.getFB()>10 || tbb.x>img2.cols ||  tbb.y>img2.rows || tbb.br().x < 1 || tbb.br().y <1){//br() bottom right坐标
21. tvalid =**false**; //too unstable prediction or bounding box out of image
22. tracked = **false**;
23. printf("Too unstable predictions FB error=%f\n",tracker.getFB());
24. **return**;
25. }
26. // 5. Estimate Confidence and Validity
27. Mat pattern;
28. Scalar mean, stdev;
29. BoundingBox bb;
30. bb.x = max(tbb.x,0);
31. bb.y = max(tbb.y,0);
32. bb.width = min(min(img2.cols-tbb.x,tbb.width),min(tbb.width,tbb.br().x));//问题：我觉得后面的min没必要呀？？
33. bb.height = min(min(img2.rows-tbb.y,tbb.height),min(tbb.height,tbb.br().y));
34. getPattern(img2(bb),pattern,mean,stdev);
35. vector<**int**> isin;
36. **float** dummy;
37. classifier.NNConf(pattern,isin,dummy,tconf); //1.tconf是用Conservative Similarity
38. tvalid = lastvalid;<span style="white-space:pre">    </span>
39. **if** (tconf>classifier.thr\_nn\_valid){//thr\_nn\_valid
40. tvalid =**true**;//2.判定轨迹是否有效，从而决定是否要增加正样本，标志位tvalid【5.6.2 P-Expert】
41. }
42. }
43. **else**
44. printf("No points tracked\n");
45. }

## ****1.Initialize points to grid****

    将bb切成10\*10的网格，将网格交点存在points，函数为TLD::bbPoints。

**[cpp]** [view plaincopy](http://blog.csdn.net/ttransposition/article/details/43195851)[在CODE上查看代码片](https://code.csdn.net/snippets/590455)

1. //将bb切成10\*10的网格，将网格交点存在points
2. **void** TLD::bbPoints(vector<cv::Point2f>& points,**const** BoundingBox& bb){
3. **int** max\_pts=10;
4. **int** margin\_h=0;//留白没有用到
5. **int** margin\_v=0;
6. **int** stepx = ceil((bb.width-2\*margin\_h)/max\_pts);//向上取整
7. **int** stepy = ceil((bb.height-2\*margin\_v)/max\_pts);
8. **for** (**int** y=bb.y+margin\_v;y<bb.y+bb.height-margin\_v;y+=stepy){
9. **for** (**int** x=bb.x+margin\_h;x<bb.x+bb.width-margin\_h;x+=stepx){
10. points.push\_back(Point2f(x,y));//最多有11\*11=121个点
11. }
12. }
13. }

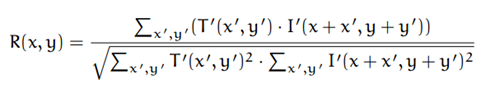
## ****2.Track points**** ****3.Estimate tracking error**** ****4.Filter out outliers****

**这三步都在函数**trackf2f 中，调用层次关系tld.processFrame->track->[tracked = tracker.trackf2f(img1,img2,points,points2)]

**[cpp]** [view plaincopy](http://blog.csdn.net/ttransposition/article/details/43195851)[在CODE上查看代码片](https://code.csdn.net/snippets/590455)

1. //points1->points2，由于调用了filterPts，所以只有通过筛选的point对还保留在points1，points2
2. **bool** LKTracker::trackf2f(**const** Mat& img1, **const** Mat& img2,vector<Point2f> &points1, vector<cv::Point2f> &points2){
3. //TODO!:implement c function cvCalcOpticalFlowPyrLK() or Faster tracking function
4. //1. Track points,Forward-Backward tracking
5. calcOpticalFlowPyrLK( img1,img2, points1, points2, status,similarity, window\_size, level, term\_criteria, lambda, 0);
6. calcOpticalFlowPyrLK( img2,img1, points2, pointsFB, FB\_status,FB\_error, window\_size, level, term\_criteria, lambda, 0);
7. //2. Estimate tracking error,Compute the real FB-error
8. **for**( **int** i= 0; i<points1.size(); ++i ){
9. FB\_error[i] = norm(pointsFB[i]-points1[i]);//残差为欧氏距离【ICPR 2】
10. }
11. //3.Filter out outliers
12. //Filter out points with FB\_error[i] > median(FB\_error) && points with sim\_error[i] > median(sim\_error)
13. normCrossCorrelation(img1,img2,points1,points2);
14. **return** filterPts(points1,points2);
15. }

    其中normCrossCorrelation(img1,img2,points1,points2)是对光流法跟踪的结果不放心，因此希望通过对比前后两点周围的小块的相似性，来进一步去掉不稳定的点。这次的相似性不是相关系数，而是normalized [cross-correlation (NCC)：](http://en.wikipedia.org/wiki/Cross-correlation)



http://images.cnitblog.com/blog/460184/201501/271556070037309.png

    这个比较复杂，建议看[wiki](http://en.wikipedia.org/wiki/Cross-correlation)的公式，其实还是前面提到的相关系数，只不过计算的时候需要自己减去均值。

**[cpp]** [view plaincopy](http://blog.csdn.net/ttransposition/article/details/43195851)[在CODE上查看代码片](https://code.csdn.net/snippets/590455)

1. **void** LKTracker::normCrossCorrelation(**const** Mat& img1,**const** Mat& img2, vector<Point2f>& points1, vector<Point2f>& points2) {
2. Mat rec0(10,10,CV\_8U);
3. Mat rec1(10,10,CV\_8U);
4. Mat res(1,1,CV\_32F);
5. **for** (**int** i = 0; i < points1.size(); i++) {
6. **if** (status[i] == 1) {//跟踪到了
7. getRectSubPix( img1, Size(10,10), points1[i],rec0 );//以points1[i]为中心，提取10\*10的小块
8. getRectSubPix( img2, Size(10,10), points2[i],rec1);
9. matchTemplate( rec0,rec1, res, CV\_TM\_CCOEFF\_NORMED);//Cross Correlation
10. similarity[i] = ((**float** \*)(res.data))[0];

13. } **else** {
14. similarity[i] = 0.0;
15. }
16. }
17. rec0.release();
18. rec1.release();
19. res.release();
20. }

    该计算的都计算好了，终于可以筛选了，filterPts(points1,points2)

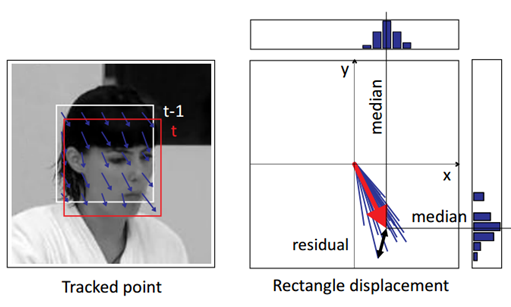
**[cpp]** [view plaincopy](http://blog.csdn.net/ttransposition/article/details/43195851)[在CODE上查看代码片](https://code.csdn.net/snippets/590455)

1. //Filter out points with FB\_error[i] > median(FB\_error) && points with sim\_error[i] > median(sim\_error)
2. **bool** LKTracker::filterPts(vector<Point2f>& points1,vector<Point2f>& points2){
3. //Get Error Medians
4. simmed = median(similarity);//NCC中值
5. **size\_t** i, k;
6. **for**( i=k = 0; i<points2.size(); ++i ){//前向筛选，没跟踪到的不要
7. **if**( !status[i])
8. **continue**;
9. **if**(similarity[i]> simmed){//normalized crosscorrelation (NCC)筛选，比对前后两点周围的小块
10. points1[k] = points1[i];
11. points2[k] = points2[i];
12. FB\_error[k] = FB\_error[i];
13. k++;
14. }
15. }
16. **if** (k==0)
17. **return** **false**;
18. points1.resize(k);
19. points2.resize(k);
20. FB\_error.resize(k);

23. fbmed = median(FB\_error);//残差中值
24. **for**( i=k = 0; i<points2.size(); ++i ){//后向筛选，找到了，但是偏离太多
25. **if**( !status[i])
26. **continue**;
27. **if**(FB\_error[i] <= fbmed){
28. points1[k] = points1[i];
29. points2[k] = points2[i];
30. k++;
31. }
32. }
33. points1.resize(k);
34. points2.resize(k);
35. **if** (k>0)
36. **return** **true**;
37. **else**
38. **return** **false**;
39. }

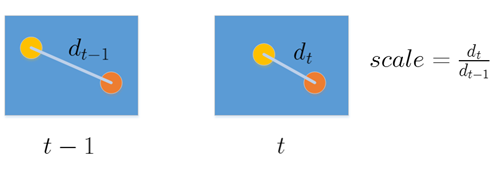
## ****5.Update bounding box****

    bbPredict(points,points2,lastbox,tbb), points和points2是前面筛选完之后的点对，现在要依据points，points2来估计bb1的**位移**和**尺度变化**，这两个信息都有了，自然可以决定lastbox在当前帧的位置tbb。



位移估计

   位移估计的方法是用所有点对x，y位移的中值作为位移的估计，如上图。尺度的估计的方法是用所有点对（同一帧）的伸缩比的中值作为尺度伸缩的估计，假设只有一堆点，尺度伸缩值的估计方式如下图：



尺度估计

**[cpp]** [view plaincopy](http://blog.csdn.net/ttransposition/article/details/43195851)[在CODE上查看代码片](https://code.csdn.net/snippets/590455)

1. //依据points1，points2估计bb1的位移和尺度变化，这两个信息都有了，自然可以决定其范围bb2
2. **void** TLD::bbPredict(**const** vector<cv::Point2f>& points1,**const** vector<cv::Point2f>& points2,
3. **const** BoundingBox& bb1,BoundingBox& bb2)    {
4. **int** npoints = (**int**)points1.size();
5. vector<**float**> xoff(npoints);
6. vector<**float**> yoff(npoints);
7. printf("tracked points : %d\n",npoints);
8. // 用位移的中值，作为目标位移的估计
9. **for** (**int** i=0;i<npoints;i++){
10. xoff[i]=points2[i].x-points1[i].x;
11. yoff[i]=points2[i].y-points1[i].y;
12. }
13. **float** dx = median(xoff);//
14. **float** dy = median(yoff);
15. **float** s;
16. // 用点对之间的距离的伸缩比例的中值，作为目标尺度变化的估计
17. **if** (npoints>1){
18. vector<**float**> d;
19. d.reserve(npoints\*(npoints-1)/2);
20. **for** (**int** i=0;i<npoints;i++){
21. **for** (**int** j=i+1;j<npoints;j++){
22. <span style="white-space:pre">            </span>  //之前比较的都是对应点之间的相似性，现在计算的是任意两点的相似性，所以更能反映拓扑结构的变化
23. <span style="white-space:pre">            </span>  //问题：假设比值是s,那么加个min(s，1/s)不是更好吗？？？？
24. <span style="white-space:pre">            </span>  //呃，好吧，亲你又YY了，这一步不是干这个的好吗？，so,这三行都忽略
25. d.push\_back(norm(points2[i]-points2[j])/norm(points1[i]-points1[j]));
26. }
27. }
28. s = median(d);//
29. }
30. **else** {
31. s = 1.0;
32. }
33. **float** s1 = 0.5\*(s-1)\*bb1.width;// top-left 坐标的偏移(s1,s2)
34. **float** s2 = 0.5\*(s-1)\*bb1.height;
35. printf("s= %f s1= %f s2= %f \n",s,s1,s2);
36. bb2.x = round( bb1.x + dx -s1);
37. bb2.y = round( bb1.y + dy -s2);
38. bb2.width = round(bb1.width\*s);
39. bb2.height = round(bb1.height\*s);
40. printf("predicted bb: %d %d %d %d\n",bb2.x,bb2.y,bb2.br().x,bb2.br().y);
41. }

## ****6.Failure detection****

    这一步很简单，原文是说A failure of the tracker is declared if http://images.cnitblog.com/blog/460184/201501/271556120036170.png pixels，其中http://images.cnitblog.com/blog/460184/201501/271556125508270.png是残差的中值，残差即反向跟踪和原始跟踪点的距离。不过程序里面还要防止目标飞到图像外面去了。

**[cpp]** [view plaincopy](http://blog.csdn.net/ttransposition/article/details/43195851)[在CODE上查看代码片](https://code.csdn.net/snippets/590455)

1. **if** (tracker.getFB()>10 || tbb.x>img2.cols || tbb.y>img2.rows || tbb.br().x < 1 || tbb.br().y <1){//br() bottom right坐标
2. tvalid =**false**; //too unstable prediction or bounding box out of image
3. tracked = **false**;
4. printf("Too unstable predictions FB error=%f\n",tracker.getFB());
5. **return**;
6. }

## ****7.Estimate Confidence and Validity****

**[cpp]** [view plaincopy](http://blog.csdn.net/ttransposition/article/details/43195851)[在CODE上查看代码片](https://code.csdn.net/snippets/590455)

1. Mat pattern;
2. Scalar mean, stdev;
3. BoundingBox bb;
4. bb.x = max(tbb.x,0);
5. bb.y = max(tbb.y,0);
6. bb.width = min(min(img2.cols-tbb.x,tbb.width),min(tbb.width,tbb.br().x));// bb.height = min(min(img2.rows-tbb.y,tbb.height),min(tbb.height,tbb.br().y));
7. getPattern(img2(bb),pattern,mean,stdev);
8. vector<**int**> isin;
9. **float** dummy;
10. classifier.NNConf(pattern,isin,dummy,tconf); //1.tconf是用Conservative Similarity
11. tvalid = lastvalid;
12. **if** (tconf>classifier.thr\_nn\_valid){//thr\_nn\_valid
13. tvalid =**true**;//2.判定轨迹是否有效，从而决定是否要增加正样本，标志位tvalid【5.6.2 P-Expert】
14. }

    注释很清楚了，大家可以先忽略判定轨迹是否有效这一部分，只要知道它是用最近邻分类器的**Conservative Similarity【5.2】**作为跟踪目标的得分即可，后面要用这个分数和检测器进行比较。

# [TLD之学习篇](http://blog.csdn.net/ttransposition/article/details/43196025)

    这一部分是TLD算法的核心之处，有了前面两篇的铺垫，终于可以毫无顾忌的说说这一部分了。不过，还有一座大山，程序的初始化，这是程序运行的铺垫，内容很多……。

## 初始化

    在**run\_tld.cpp**中，一旦你选的要跟踪的目标box后便会调用初始化init:

tld.init(last\_gray,box,bb\_file);//初始目标位置存储在box

内容太多，大伙看下面程序中标出的9点吧，其中3、5、8三点已经讲过啦，所调用的函数就不再展开，具体见我共享的注释，后面再补充介绍几点：

**[cpp]** [view plaincopy](http://blog.csdn.net/ttransposition/article/details/43196025)[在CODE上查看代码片](https://code.csdn.net/snippets/590494)

1. **void** TLD::init(**const** Mat& frame1,constRect&box,**FILE**\*bb\_file){
2. //bb\_file =fopen("bounding\_boxes.txt","w");
3. //Get Bounding Boxes
4. // 1.预先计算好所有位置，所有尺度的bb,并计算每一个bb与初始跟踪区域的面积交/并
5. // 问题：好多呀，有必要吗？一劳永逸的事情，倒也耽误不了多少工夫
6. buildGrid(frame1,box);
7. printf("Created %dbounding boxes\n",(**int**)grid.size());
8. //Preparation
9. //allocation
10. iisum.create(frame1.rows+1,frame1.cols+1,CV\_32F);
11. iisqsum.create(frame1.rows+1,frame1.cols+1,CV\_64F);
12. dconf.reserve(100);
13. dbb.reserve(100);
14. bbox\_step =7;
15. //tmp.conf.reserve(grid.size());
16. tmp.conf = vector<**float**>(grid.size());
17. tmp.patt = vector<vector<**int**>>(grid.size(),vector<**int**>(10,0));
18. //tmp.patt.reserve(grid.size());
19. dt.bb.reserve(grid.size());//预留空间
20. good\_boxes.reserve(grid.size());
21. bad\_boxes.reserve(grid.size());
22. pEx.create(patch\_size,patch\_size,CV\_64F);
23. //Init Generator
24. generator = PatchGenerator(0,0,noise\_init,**true**,1-scale\_init,1+scale\_init,-angle\_init\*CV\_PI/180,angle\_init\*CV\_PI/180,-angle\_init\*CV\_PI/180,angle\_init\*CV\_PI/180);
25. // 2. 依据1中计算的overlap,选出10(num\_closest\_init)个good box，1个best\_box,其余的都作为bad box（N个）
26. //【5.6.1】select10 bounding boxes on the  scanning gridthat are closest to the initial bounding box.
27. getOverlappingBoxes(box,num\_closest\_init);
28. printf("Found %d goodboxes, %d bad boxes\n",(**int**)good\_boxes.size(),(**int**)bad\_boxes.size());
29. printf("Best Box: %d%d %d %d\n",best\_box.x,best\_box.y,best\_box.width,best\_box.height);
30. printf("Bounding boxhull: %d %d %d %d\n",bbhull.x,bbhull.y,bbhull.width,bbhull.height);
31. //Correct BoundingBox
32. lastbox=best\_box;//注意是best\_box
33. lastconf=1;
34. lastvalid=**true**;
35. fprintf(bb\_file,"%d,%d,%d,%d,%f\n",lastbox.x,lastbox.y,lastbox.br().x,lastbox.br().y,lastconf);
36. //Prepare Classifier
37. //3. 确定随机森林分类器特征的计算方式，随机产生了 130对点，比较其相互的大小关系（二值化）->作为10个列向量（向量化）
38. classifier.prepare(scales);
39. ///Generate Data
40. // Generate positivedata【5.6.1】
41. // generate 20warped versions by geometric transformations(shift \_1%,scale change\_1%,in-plane rotation 10)
42. // and add them withGaussian noise(\_ ¼ 5) on pixels.The result is 200 synthetic positive patches.
43. // 4. 得到最近邻分类器的正样本pEx，提取的是best\_box的patch;随机森林分类器的正样本由 goodbox 变形繁衍（1->20）得到
44. generatePositiveData(frame1,num\_warps\_init);
45. // 5. Set variancethreshold
46. Scalar stdev,mean;
47. meanStdDev(frame1(best\_box),mean,stdev);//注意是best\_box，而不是我们自己框定的box
48. integral(frame1,iisum,iisqsum);
49. var = pow(stdev.val[0],2)\*0.5; //【5.3.1】50percent of variance of the patch that was selected  for tracking
50. cout << "variance:" << var << endl;
51. //check variance
52. **double** vr=  getVar(best\_box,iisum,iisqsum)\*0.5;
53. cout << "checkvariance: " << vr << endl;
54. // 6. Generatenegative data，得到随机森林的负样本集:nX（特征fern，N多），最近邻的负样本集：nEx（图像块patch,100个）
55. generateNegativeData(frame1);
56. // 7. 构造训练集和测试集（负样本五五分）
57. //Split NegativeFerns into Training and Testing sets (they are already shuffled)
58. **int** half =(**int**)nX.size()\*0.5f;//保留后一半作为测试
59. nXT.assign(nX.begin()+half,nX.end());
60. nX.resize(half);
61. //Split Negative NNExamples into Training and Testing sets
62. half = (**int**)nEx.size()\*0.5f;
63. nExT.assign(nEx.begin()+half,nEx.end());
64. nEx.resize(half);
65. //Merge NegativeData with Positive Data and shuffle it
66. // [nX +pX]-->ferns\_data
67. vector<pair<vector<**int**>,**int**>> ferns\_data(nX.size()+pX.size());
68. vector<**int**> idx= index\_shuffle(0,ferns\_data.size());//再一次打乱+-样本的顺序
69. **int** a=0;
70. **for** (inti=0;i<pX.size();i++){//pX是在 generatePositiveData中产生 10\*20
71. ferns\_data[idx[a]] = pX[i];
72. a++;
73. }
74. **for** (inti=0;i<nX.size();i++){
75. ferns\_data[idx[a]] = nX[i];
76. a++;
77. }
78. //Data already havebeen shuffled, just putting it in the same vector
79. //[pEx(1个) nEx(N多)]->nn\_data
80. vector<cv::Mat>nn\_data(nEx.size()+1);
81. nn\_data[0] = pEx;
82. **for** (inti=0;i<nEx.size();i++){
83. nn\_data[i+1]= nEx[i];
84. }
85. // 8.Training，决策森林和最近邻
86. classifier.trainF(ferns\_data,2); //bootstrap= 2
87. classifier.trainNN(nn\_data);
88. // 9.ThresholdEvaluation on testing sets，检查是否要提高阈值thr\_fern，thr\_nn，thr\_nn\_valid
89. classifier.evaluateTh(nXT,nExT);//仅此一次调用，后面都不会提高了
90. }

### 1. buildGrid

    前面说过TLD是**多尺度滑动窗口检测**，所以这就把所有窗口能滑到的bb（包括坐标空间，尺度空间）都计算好啦，存在**grid**里面。

### 7. 构造训练集和测试集

|  |  |  |
| --- | --- | --- |
|  | 训练集 | 测试集（都只有负样本） |
| NN分类器 | [pEx(1个) nEx(N/2)]-->nn\_data | nExT （N/2） |
| 随机森林 | [nX(N/2) + pX(200)]-->ferns\_data | nXT(N/2) |

### 4.generatePositiveData【5.6.1】

    这个函数很有争议，[很多人](http://quandb2007.blog.163.com/blog/static/4187887520135873314523/)认为程序错了，其实不然。各位看客可以先自己试着找找，看看能否找到错误。

**[cpp]** [view plaincopy](http://blog.csdn.net/ttransposition/article/details/43196025)[在CODE上查看代码片](https://code.csdn.net/snippets/590494)

1. **void** TLD::generatePositiveData(**const** Mat& frame, intnum\_warps){
2. Scalar mean;
3. Scalar stdev;
4. getPattern(frame(best\_box),pEx,mean,stdev);//pEx
5. //Get Fern featureson warped patches
6. Mat img;
7. Mat warped;
8. GaussianBlur(frame,img,Size(9,9),1.5);
9. warped = img(bbhull);
10. RNG&rng = theRNG();
11. Point2fpt(bbhull.x+(bbhull.width-1)\*0.5f,bbhull.y+(bbhull.height-1)\*0.5f);//水平，垂直中心，即旋转的中心
12. vector<**int**>fern(classifier.getNumStructs());
13. pX.clear();//
14. Mat patch;
15. **if**(pX.capacity()<num\_warps\*good\_boxes.size())
16. pX.reserve(num\_warps\*good\_boxes.size());
17. **int** idx;
18. **for** (inti=0;i<num\_warps;i++){//每一个good\_boxes都生num\_warps个pX
19. **if**(i>0)
20. generator(frame,pt,warped,bbhull.size(),rng);//仿射变换    for (intb=0;b<good\_boxes.size();b++){
21. idx=good\_boxes[b];
22. patch = img(grid[idx]);
23. classifier.getFeatures(patch,grid[idx].sidx,fern);
24. pX.push\_back(make\_pair(fern,1));
25. }
26. }
27. printf("Positiveexamples generated: ferns:%d NN:1\n",(**int**)pX.size());
28. }

     揭晓答案，有些人认为

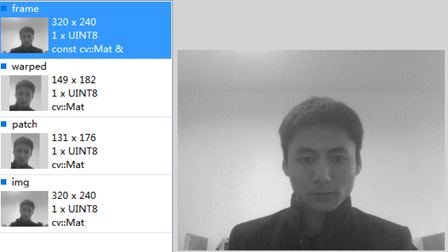
generator(frame,pt,warped,bbhull.size(),rng);

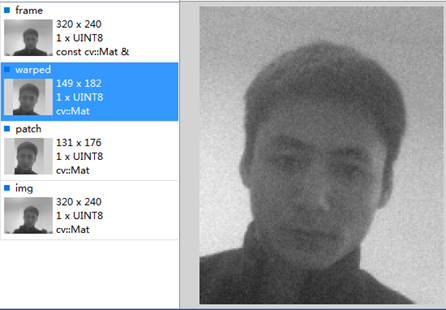
    这一步错了，因为仿射变换的结果是放在warped里面，而后来还是从img提取patch（patch = img(grid[idx])），所以他们觉得应该修改成：

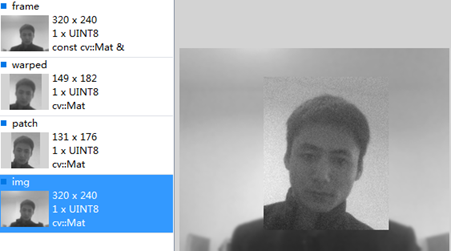
generator(img,pt,img,frame.size(),rng);

    开始我也认为错了，三人成虎呀。事实真的如此吗？其实，我们忽略了一个细节，注意到  warped = img(bbhull); 也就是说warped是img bbhull区域的浅拷贝，它俩是共享存储空间的，于是warped一变，img也就会变。

本着为科学真理献身的精神，我用[ImageWatch](http://opencv.org/image-watch-plugin-for-visual-studio.html)这个工具给自己做了个验证实验……，为了写个笔记，我也是蛮拼的……







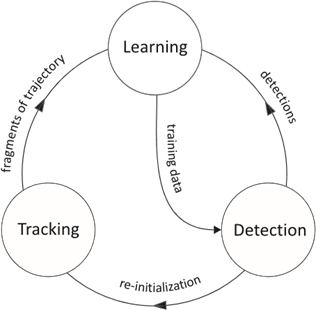
    这是generator执行之后的结果，可以看到frame没变，是原始帧的图像，很明显，warped沿顺时针反向旋转了，接下来，我们惊奇地发现，img的bbhull部分也旋转了，有图有真相，啥也不用说了吧。

    我没有找到这个函数的API说明，所以各位不妨通过上面三张图，再猜猜generator(frame,pt,warped,bbhull.size(),rng)的参数含义和具体实现，frame肯定是输入图像，pt是旋转中心，warped自然是输出结果，而bbhull.size()是最终要保留的区域大小，rng自然是仿射变换的参数。我觉得算法可能是先将整个frame以pt为中心，进行仿射变换，将结果暂存，假设是放在temp中，然后再从temp提取出bbhull区域。哈哈，这样做自然是做了很多无用功。

    现在再回头看看generator(img,pt,img,frame.size(),rng)这种改法，其实并不可取，原来每次都是使用frame，做一点点变化得到正样本，而改成这样后，每次就都是在上一次的基础上继续变化，所以20次变化后，img和原始frame可能就相去甚远，比如，很不巧，每次都顺时针转9°，20次变化之后，我就完全倒立了……。那么这个样本有必要加到训练集吗，No！

## 程序流程

    当我们框定要跟踪的目标后，会调用回调函数，mouseHandler，然后就是前面介绍的tld.init(last\_gray,box,bb\_file)，之后的工作就很枯燥了，循环处理每一帧图像 tld.processFrame(last\_gray,current\_gray,pts1,pts2,pbox,status,tl,bb\_file);



    processFrame函数里面就是T、L、D三大主角要上演的剧本啦，没想到前的主角介绍，场景搭建工作这么费事……，哎，写博客真是一项苦力活。下面这段程序，各位看客仔细看看，最好能理解每一步是在做什么。

**[cpp]** [view plaincopy](http://blog.csdn.net/ttransposition/article/details/43196025)[在CODE上查看代码片](https://code.csdn.net/snippets/590494)

1. **void** TLD::processFrame(**const** cv::Mat&img1,**const** cv::Mat&img2,vector<Point2f>&points1,vector<Point2f>&points2,BoundingBox&bbnext,**bool**&lastboxfound, booltl, **FILE**\*bb\_file){
2. vector<BoundingBox>cbb;//聚类之后的bounding box
3. vector<**float**>cconf;
4. intconfident\_detections=0;//小D的结果聚类之后，分数比小T高的数目
5. **int** didx; //detectionindex
6. /// 1.Track
7. **if**(lastboxfound&&tl){//前一帧目标出现过，我们才跟踪，否则只能检测了
8. track(img1,img2,points1,points2);
9. }
10. **else**{
11. tracked = **false**;
12. }
13. /// 2.Detect
14. detect(img2);
15. /// 3.Integration
16. **if**(tracked){
17. bbnext=tbb;//  小T， bbnext这是你下一次要跟踪的目标
18. lastconf=tconf;
19. lastvalid=tvalid;
20. printf("Tracked\n");
21. //--------- 4. Detetor Vs Trackr-------------
22. **if**(detected)
23. { //   if Detected
24. clusterConf(dbb,dconf,cbb,cconf);//检测的结果太多，所以要进行非极大值抑制，这里是用cluster的方法，时间消耗应该不少吧？？
25. printf("Found %dclusters\n",(**int**)cbb.size());
26. //Get index of a clusters that is farfrom tracker and are more confident than the tracker
27. **for** (inti=0;i<cbb.size();i++){
28. **if**(bbOverlap(tbb,cbb[i])<0.5 && cconf[i]>tconf){//小D小T分歧很大，而且小D更有把握（都是用Conservativesimilarity对比）
29. confident\_detections++;//看看小D是不是眼花了
30. didx=i; //detectionindex
31. }
32. }
33. /\*----------------小T向小D学习------------------\*/
34. **if**(confident\_detections==1){//小D没有眼花，而且看得比小T更清楚
35. printf("Found a bettermatch..reinitializing tracking\n");//if there is ONEsuch a cluster, re-initialize the tracker
36. bbnext=cbb[didx];//重新初始化要跟踪的目标
37. lastconf=cconf[didx];//
38. lastvalid=**false**; // 小T，你这一帧表现不好，所以这次小D就不跟你学习了
39. }
40. **else** {
41. printf("%d confidentcluster was found\n",confident\_detections);
42. intcx=0,cy=0,cw=0,ch=0;
43. intclose\_detections=0;
44. **for** (**int** i=0;i<dbb.size();i++){
45. **if**(bbOverlap(tbb,dbb[i])>0.7){ // Get mean of close detections
46. cx +=dbb[i].x;
47. cy +=dbb[i].y;
48. cw +=dbb[i].width;
49. ch +=dbb[i].height;
50. close\_detections++;
51. printf("weighteddetection: %d %d %d %d\n",dbb[i].x,dbb[i].y,dbb[i].width,dbb[i].height);
52. }
53. }
54. **if**(close\_detections>0){
55. bbnext.x =cvRound((**float**)(10\*tbb.x+cx)/(**float**)(10+close\_detections));// weighted averagetrackers trajectory with the close detections
56. bbnext.y =cvRound((**float**)(10\*tbb.y+cy)/(**float**)(10+close\_detections));
57. bbnext.width =cvRound((**float**)(10\*tbb.width+cw)/(**float**)(10+close\_detections));
58. bbnext.height=  cvRound((**float**)(10\*tbb.height+ch)/(**float**)(10+close\_detections));
59. printf("Trackerbb: %d %d %d %d\n",tbb.x,tbb.y,tbb.width,tbb.height);
60. printf("Average bb: %d%d %d %d\n",bbnext.x,bbnext.y,bbnext.width,bbnext.height);
61. printf("Weighting%d close detection(s) with tracker..\n",close\_detections);
62. }
63. **else**{
64. printf("%dclose detections were found\n",close\_detections);
65. }
66. }
67. }
68. }
69. **else**{                                      //   If NOT tracking
70. printf("Nottracking..\n");
71. lastboxfound = **false**;
72. lastvalid = **false**;
73. **if**(detected){                          //  and detector is defined
74. clusterConf(dbb,dconf,cbb,cconf);  //  cluster detections
75. printf("Found%d clusters\n",(**int**)cbb.size());
76. **if**(cconf.size()==1){//大于1呢？，眼花了呗
77. bbnext=cbb[0];//注意使用cbb来初始化
78. lastconf=cconf[0];
79. printf("Confidentdetection..reinitializing tracker\n");
80. lastboxfound = **true**;
81. }
82. }
83. }
84. lastbox=bbnext; //
85. **if** (lastboxfound)
86. fprintf(bb\_file,"%d,%d,%d,%d,%f\n",lastbox.x,lastbox.y,lastbox.br().x,lastbox.br().y,lastconf);
87. **else**
88. fprintf(bb\_file,"NaN,NaN,NaN,NaN,NaN\n");
89. **if**(lastvalid && tl)//tvalid
90. //------------- 5. learn ---------------
91. learn(img2);
92. }

    首先我觉得大家自己应该试着想一下，检测结果的可能出现的各种情况，以及该怎么处理。

    下面是我列出来的情况，以及作者是怎么处理的，算是对作者写这一部分代码时现场的还原吧，哈哈，有没有感觉像侦探在还原犯罪现场，so，不保证完全正确。

    （1）英雄所见略同，二者检测的目标区域很相似

条件，bbOverlap(tbb,cbb[i])>=0.5对所有cbb都满足，就没必要你争我抢的了，**最终结果应该是综合小D和小T，可是作者是以小T为主，即小T参考小D的检测结果，表示不太公平，不过貌似也没有什么综合的好法子，能够平等对待小D，再说小T能跟踪到的目标，就已经表明这个目标具有帧与帧直接的连续性，是相对比较靠谱的**。这种情况下分数一般还是会有差距的，有差距就说明存在学习的空间呀，所以相互学习一下也是极好的。

    1）如果小T的检测分数更高，那么还是让**小D向小T学习下吧**，虽说小D的检测结果已经比较好了，这样做的必要性不是很大，但是我们要精益求精嘛。

    2）如果小D的检测分数更高，**作者并没有让小T学习小D，相反，是让小D学习小T，这个地方貌似有点不近情理**。小T不用学习倒是可以理解，毕竟跟踪器需要学习的就是目标的新位置，而它的位置已经比较接近了，所以学习就显得没那么必要了。但是让小D学习小T，表示很郁闷，如果说小T的分数更低呢？你这不是误导小D吗？哈哈，这个就要到学习部分就知道了，剧透一下，小D学不学习小T还得过一关。我觉得，作者让小D学习小T的初衷，应该还是希望小D能够精益求精，但是对待小T可就没这么严格了。

    （2）小D眼瞎了……好吧，这个词的不太恰当，总之，小D没有检测到目标，也就是说，所有grid里面的bb都没通过小D的三道屏障。最后的**结果自然是小T说了算，小D还得向小T虚心学习**，这种情况可能是待跟踪的目标的外形发生了改变，比如遮挡。

    （3）小D眼花了

    这次小D检测到了目标，不过可疑的目标很多，但是TLD 1.0是单目标跟踪算法，如果出现多个可疑目标的话，那么检测结果自然是有问题的，判断方法：看if (confident\_detections>1)是否成立了。

前面说过，虽然小D眼花了，但长兄为父，所以，最终结果还是得**参考**小T**，即小T参考小D的检测结果，当然，小D也要向小T虚心学习**。

    （4）小D的结果比小T差

    小D的分数都没小T高，cconf[i]<=tconf，那么confident\_detections=0，**处理方式同上。**

    （5）小D结果比小T好，而且双方的分歧很大

    判断方法：if(confident\_detections==1),这种情况下，**小T要向小D学习**，霸气的**小D自然是不需要向小T学习**，**最终的结果由小D说了算**。

    其实，还有很多种情况的，比如，小T没跟踪到（可依据小D的结果，可再细分），小T和小D都没跟踪到。

    总之，这些情况下要做的事情主要是这三件：

* 综合小D小T的检测结果
* 小T向小D学习
* 小D向小T学习

## 小T参考小D的检测结果

    注意，我用的是参考,所以有可能不理会小D，何时参考呢？，依据原则是bbOverlap(tbb,dbb[i])>0.7是否成立。

    1）小D的结果和小T比较接近：bbOverlap(tbb,dbb[i])>0.7

    2）小D错的离谱，还自以为是：bbOverlap(tbb,dbb[i])<=0.7

    具体参考方式：对bbOverlap(tbb, dbb[i])>0.7的dbb[i]加权求重心，宽和高，tbb的权重为10,而小D的权重为1，权重如此悬殊，是因为小T每次只能得到一个bb，而小D每次能得到很多个。

**[cpp]** [view plaincopy](http://blog.csdn.net/ttransposition/article/details/43196025)[在CODE上查看代码片](https://code.csdn.net/snippets/590494)

1. **for** (**int** i = 0; i<dbb.size(); i++){
2. **if**(bbOverlap(tbb, dbb[i])>0.7){ // Getmean of close detections
3. cx += dbb[i].x;
4. cy += dbb[i].y;
5. cw += dbb[i].width;
6. ch += dbb[i].height;
7. close\_detections++;
8. printf("weighteddetection: %d %d %d %d\n", dbb[i].x, dbb[i].y, dbb[i].width, dbb[i].height);
9. }
10. }
11. **if** (close\_detections>0){
12. bbnext.x = cvRound((**float**)(10 \*tbb.x + cx) / (**float**)(10 + close\_detections));// weighted average trackers trajectory with theclose detections
13. bbnext.y = cvRound((**float**)(10 \*tbb.y + cy) / (**float**)(10 + close\_detections));
14. bbnext.width = cvRound((**float**)(10 \*tbb.width + cw) / (**float**)(10 + close\_detections));
15. bbnext.height = cvRound((**float**)(10 \*tbb.height + ch) / (**float**)(10 + close\_detections));
16. printf("Tracker bb: %d%d %d %d\n", tbb.x, tbb.y, tbb.width, tbb.height);
17. printf("Average bb: %d%d %d %d\n", bbnext.x, bbnext.y, bbnext.width, bbnext.height);
18. printf("Weighting %dclose detection(s) with tracker..\n", close\_detections);
19. }

## 小T向小D学习

    我觉得看这一段的注释已经够直白了，不细说了，重点是bbnext=cbb[didx];//重新初始化要跟踪的目标。

**[cpp]** [view plaincopy](http://blog.csdn.net/ttransposition/article/details/43196025)[在CODE上查看代码片](https://code.csdn.net/snippets/590494)

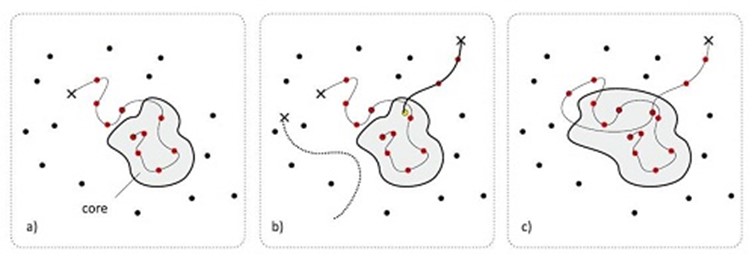
1. **if** (confident\_detections==1){ //小D没有眼花，而且看得比小T更清楚
2. printf("Found a bettermatch..reinitializing tracking\n");//if there is ONEsuch a cluster, re-initialize the tracker
3. bbnext=cbb[didx];//重新初始化要跟踪的目标
4. lastconf=cconf[didx];//
5. lastvalid=**false**; // 小T，你这一帧表现不好，所以这次小D就不跟你学习了
6. }

小D向小T学习

   这一部分内容稍微多点，首先要明白小D之所以要学习，原因有两方面：

    （1）更新检测器

    因为目标的外形可能会不断变化，那么小D需要及时将这些变化的样本加入训练集重新训练。问题来了，目标外形变化之后，小D自然是很难检测到的，那谁来告我们哪些是要加入的新样本呢？好吧，看标题就知道，自然是小T，问题又来了，小T是能利用目标在帧与帧之间的连续性，跟踪到小D所不能检测到的目标，可是小T的结果不一定靠谱呀，如果小T错了，岂不导致小D也跟着错……。所以，我们还得甄别小T的结果是否靠谱，这一步很困难【5.6.2】。



    作者的方法看起来有点玄乎，其实只是最近邻分类器来判断，不过这个最近邻分类器的样本点加了时间限制，按照正样本加入的时间顺序，只保留前50%的正样本。图中，黑色点是负样本，红色点是正样本，**都是特征空间！不是坐标空间**！，×是轨迹的起点，然后就可以依据前50%的正样本点，得出最近邻的分界线，也就是图中黑色的圈圈。当跟踪的目标其在特征空间属于这个core区域就认为是有效的轨迹。这一部分实现在TLD::track的最后一步

**[cpp]** [view plaincopy](http://blog.csdn.net/ttransposition/article/details/43196025)[在CODE上查看代码片](https://code.csdn.net/snippets/590494)

1. **if**(tconf>classifier.thr\_nn\_valid){//thr\_nn\_valid
2. tvalid =**true**;//2.判定轨迹是否有效，从而决定是否要增加正样本，标志位tvalid【5.6.2P-Expert】
3. }

   既然轨迹是有效的，那么就认为这个样本是真正的正样本，于是和初始化类似，取周围10个最接近的bb,然后仿射变换由1->10，最后能得到100个正样本。这一部分和下面部分的代码实现都在TLD::learn中，后面一并给出。

    这一部分是为检测器样本输送比较靠谱的正样本，也就是原文说的P-expert。

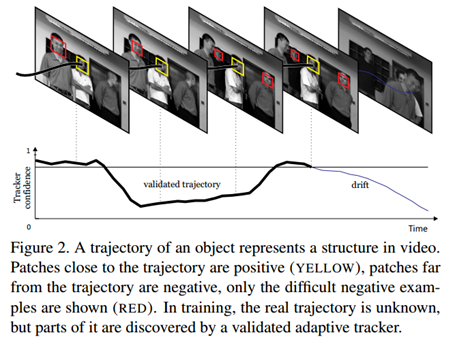
P-expert exploitsthe temporal structure in the video and assumes that the object moves along atrajectory. The P-expert remembers the location of the object in the previousframe and estimates the object location in current frame using a frame-to-frametracker. If the detector labeled the current location as negative (i.e., madefalse negative error), the P-expert generates a positive example.

    （2）校正检测器

    如果检测器的结果不如跟踪器，那就说明检测器有可能错了，有错自然要改。首先我们要清楚，何时有错，由于是单目标跟踪，于是只能有一个正确的目标，**当跟踪器的结果靠谱的时候，那么目标周围的窗口就都是负样本**（当然还是得保持一点距离，于是原文认为与目标面积的交/并<0.2的窗口都是负样本），直觉上，我们觉得这样肯定是有风险的，到底有没有风险呢？参见原文4.2 节。

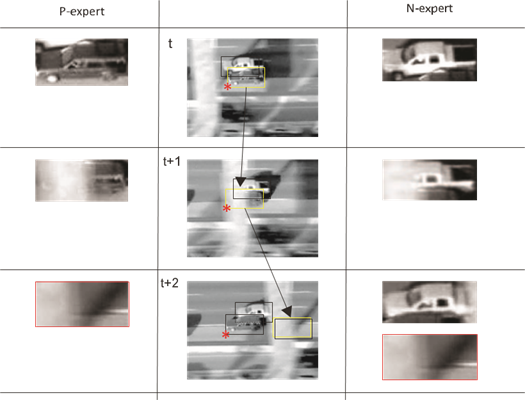
    这里再详细介绍一下负样本。我们的目的是要用新的负样本训练分类器，但是，我们都知道用hard negative训练更为有效，于是我们还要对负样本进行筛选，对于**随机森林分类器**而言，只有能通过第一关（方差）的负样本才有必要加入它的训练集(其实只是统计直方图)。对于最**近邻分类器**而言，只有连续通过前两关的负样本才有资格作为最近邻的负样本，不过，对于那些还能被检测器误认为是正样本的检测窗口来说，将它们作为负样本，效果肯定更好，而且最近邻分类器的分类时间直接和样本数目成正比例关系，所以也负担不起太多负样本呀，于是作者就这么干了【5.6.3 N-Expert】。

N-expert exploitsthe spatial structure in the video and assumes that the object can appear at asingle location only.The N-expert analyzes all responses of the detector in thecurrent frame and the response produced by the tracker and selects the one thatis the most confident. Patches that are not overlapping with the maximallyconfident patch are labeled as negative. The maximally confident patchreinitializes the location of the tracker.



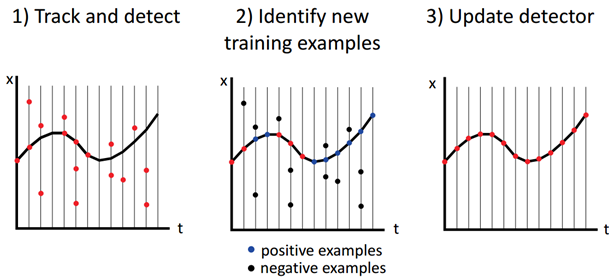
P-N Learning:Bootstrapping Binary Classifiers by Structural Constraints CVPR 2010

    上图中红色bb即依据track是否有效而添加的hard negative。



   上图就是一个校正检测器的例子，带\*号的是N-Expert认为的目标区域，黄色bb是P-expert所认为的目标区域，t+2时，P-expert错了，而N-Expert及时发现了这一错误，那么下一次训练的时候就可以校正了。这张图，**我认为和作者的P-expert ，N-Expert是可以达到这个目的，但是他的代码实现并不能达到这个目的**。因为作者的P-expert，N-Expert都是建立在跟踪轨迹有效的基础上，那么目标区域只有一个，而图中是有两个，所以必定也要给N-Expert赋予能找到目标的能力，而代码实现部分，他只有找负样本的能力。**So,这一部分我很迷惑，希望得到高人指点。**

   最后，再看一个一维的图示。



   注意看上面的图，红色点应该是检测器的输出，结合跟踪的轨迹之后，与轨迹不符合的红色点被标成了黑色，即负样本，对应检测器的校正，而蓝色点是新增的正样本，即对应检测器的更新。当然，这是比较理想的情况，跟踪器一直比较稳定，而检测器不是很稳定。

## TLD::learn函数

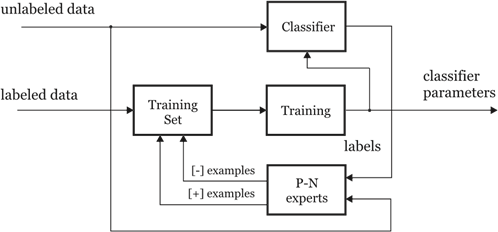
   对照前面的内容，这段程序应该说得比较清楚了吧.

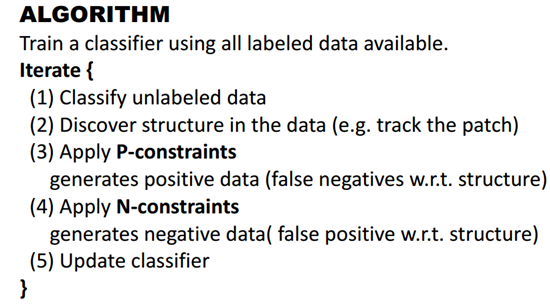
**[cpp]** [view plaincopy](http://blog.csdn.net/ttransposition/article/details/43196025)[在CODE上查看代码片](https://code.csdn.net/snippets/590494)

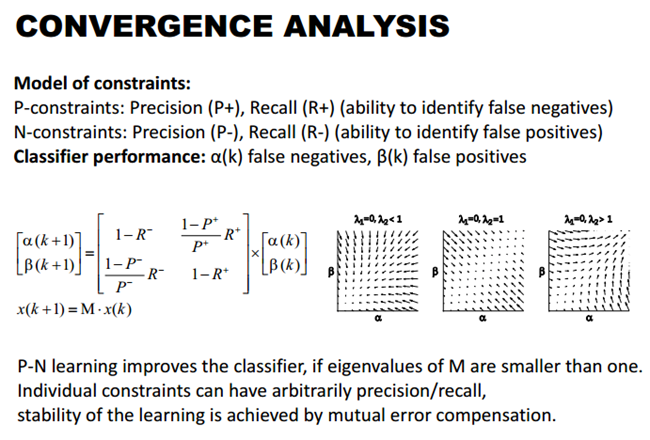
1. **void** TLD::learn(**const** Mat& img){// current\_gray
2. printf("[Learning]");
3. ///Check consistency
4. BoundingBox bb;
5. bb.x = max(lastbox.x,0); //lastbox
6. bb.y = max(lastbox.y,0);
7. bb.width = min(min(img.cols-lastbox.x,lastbox.width),min(lastbox.width,lastbox.br().x));
8. bb.height = min(min(img.rows-lastbox.y,lastbox.height),min(lastbox.height,lastbox.br().y));
9. Scalar mean,stdev;
10. Matpattern;
11. getPattern(img(bb),pattern,mean,stdev);//pattern：resizedZero-Mean patch，为什么要弄成0均值呢？是算相关系数
12. vector<**int**>isin;
13. **float** dummy,conf;
14. // 1. 再粗略地检测一遍lastbox，因为结果是加权的，如果偏差很大，岂不是误导检测器
15. classifier.NNConf(pattern,isin,conf,dummy);
16. **if**(conf<0.5) {//Relative Similarity，注意:nn\_thr >= 0.65，所以阈值降低了，因为我们要迎接新人
17. printf("Fastchange..not training\n");//形变是缓慢的，你如此不同，应该不是同类
18. lastvalid =**false**;
19. **return**;
20. }
21. **if**(pow(stdev.val[0],2)<var){
22. printf("Lowvariance..not training\n");
23. lastvalid=**false**;
24. **return**;
25. }
26. **if**(isin[2]==1){//是否是负样本
27. printf("Patchin negative data..not traing");
28. lastvalid=**false**;
29. **return**;
30. }
31. /// Data generation
32. **for** (inti=0;i<grid.size();i++){//为getOverlappingBoxes函数预先计算grid[i].overlap
33. grid[i].overlap =bbOverlap(lastbox,grid[i]);
34. }
35. vector<pair<vector<**int**>,**int**>> fern\_examples;
36. good\_boxes.clear();
37. bad\_boxes.clear();
38. // 2. 用lastbox，重新计算good,bad,bestbb还有 bbhull
39. getOverlappingBoxes(lastbox,num\_closest\_update);//num\_closest\_update： 10
40. **if**(good\_boxes.size()>0)
41. // 3. 更新这一帧的 pX pEx,【5.6.2P-Expert】
42. generatePositiveData(img,num\_warps\_update);//注意：是用best\_box，而不是lastbox
43. **else**{
44. lastvalid = **false**;
45. printf("No goodboxes..Not training");
46. **return**;
47. }
48. fern\_examples.reserve(pX.size()+bad\_boxes.size());
49. fern\_examples.assign(pX.begin(),pX.end());
50. **int** idx;
51. // 4. 从bad\_boxes挑选hardnegative作为新增的随机森林训练负样本集【5.6.3N-Expert】
52. **for** (inti=0;i<bad\_boxes.size();i++){
53. idx=bad\_boxes[i];
54. **if**(tmp.conf[idx]>=1){//回忆一下 grid->方差->结果存在tmp，conf是随机森林的分数
55. fern\_examples.push\_back(make\_pair(tmp.patt[idx],0));
56. }
57. }
58. // 5. 从dt.bb中挑选hardnegative作为新增的最近邻分类器的负样本集【5.6.3N-Expert】
59. vector<Mat> nn\_examples;
60. nn\_examples.reserve(dt.bb.size()+1);
61. nn\_examples.push\_back(pEx);//唯一一个正样本
62. **for** (inti=0;i<dt.bb.size();i++){
63. idx = dt.bb[i];
64. **if**(bbOverlap(lastbox,grid[idx]) < bad\_overlap)
65. nn\_examples.push\_back(dt.patch[i]);
66. }
67. /// 6. Classifiersupdate
68. classifier.trainF(fern\_examples,2);
69. classifier.trainNN(nn\_examples);
70. //问题：fern\_examples和nn\_examples都是新的数据，完全没有用到之前的pX，pEx,nx,nEx？？？
71. //原来，随机森林分类器只要保存直方图统计就可以了，所以不需要存储正负样本集
72. //而最近邻分类器，并没有clearpEx,nEx,而是不断地追加正负样本
73. classifier.show();
74. }

上升到P-NLearning

    Ref：P-N Learning Bootstrapping Binary Classifiers by StructuralConstraints CVPR 10







    这一部分以后再写，准备回家……