# Tracking-Learning-Detection原理分析

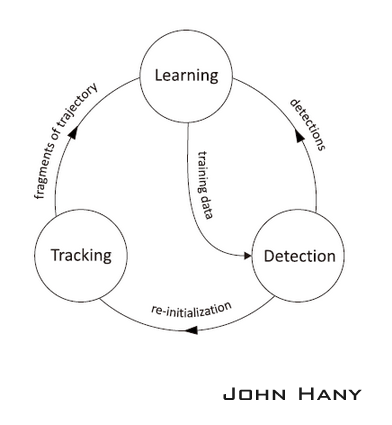
        Tracking-Learning-Detection（TLD）是[Zdenek Kalal](http://personal.ee.surrey.ac.uk/Personal/Z.Kalal/index.html)提出的一种对视频中单个物体长时间跟踪的算法。我主要会根据他在2010年发表的论文《Tracking-Learning-Detection》来分析TLD算法的原理。该项目的[首页](http://personal.ee.surrey.ac.uk/Personal/Z.Kalal/tld.html)中有几段视频展示了TLD实时跟踪的效果和性能，其中的两个视频可以在这里下载：[similar objects](http://pan.baidu.com/s/1eQh6J9S)，[human face](http://pan.baidu.com/s/1dDioJIx)。作者公布了[源代码](https://github.com/zk00006/OpenTLD)，不过需要Matlab和Visual Studio交叉编译，在我的机器上没能运行。GitHub上有很多C++版本的TLD，比如[arthurv](https://github.com/arthurv/OpenTLD)，注释比较详细，但速度很慢。

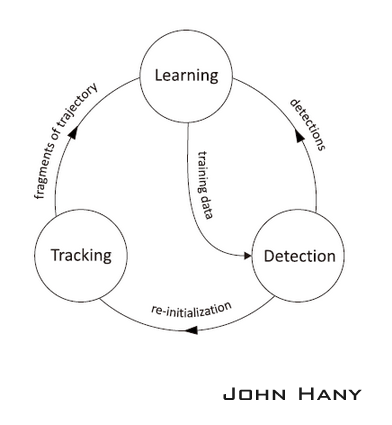
*由于我使用的插件默认对所有图片加水印，所以本文中使用的大多数并不是我绘制的图片也加上了水印，还请谅解。*

        正如名字所示，TLD算法主要由三个模块构成：**追踪器**（tracker），**检测器**（detector）和**机器学习**（learning）。

        对于视频追踪来说，常用的方法有两种，一是使用**追踪器**根据物体在上一帧的位置预测它在下一帧的位置，但这样会积累误差，而且一旦物体在图像中消失，追踪器就会永久失效，即使物体再出现也无法完成追踪；另一种方法是使用**检测器**，对每一帧单独处理检测物体的位置，但这又需要提前对检测器离线训练，只能用来追踪事先已知的物体。

        TLD是对视频中未知物体的长时间跟踪的算法。“未知物体”指的是任意的物体，在开始追踪之前不知道哪个物体是目标。“长时间跟踪”又意味着需要算法实时计算，在追踪中途物体可能会消失再出现，而且随着光照、背景的变化和由于偶尔的部分遮挡，物体在像素上体现出来的“外观”可能会发生很大的变化。从这几点要求看来，单独使用追踪器或检测器都无法胜任这样的工作。所以作者提出把追踪器和检测器结合使用，同时加入机器学习来提高结果的准确度。

[](http://johnhany.net/2014/05/tld-the-theory/#image-568)

[[](http://johnhany.net/2014/05/tld-the-theory/#imageclose-568)](http://johnhany.net/2014/05/tld-the-theory/" \l "imageclose-568)

**追踪器**的作用是跟踪连续帧间的运动，当物体始终可见时跟踪器才会有效。追踪器根据物体在前一帧已知的位置估计在当前帧的位置，这样就会产生一条物体运动的轨迹，从这条轨迹可以为学习模块产生正样本（Tracking->Learning）。

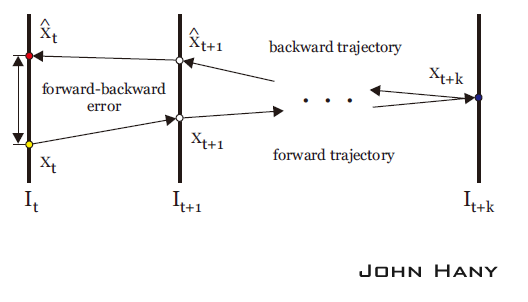
**检测器**的作用是估计追踪器的误差，如果误差很大就改正追踪器的结果。检测器对每一帧图像都做全面的扫描，找到与目标物体相似的所有外观的位置，从检测产生的结果中产生正样本和负样本，交给学习模块（Detection->Learning）。算法从所有正样本中选出一个最可信的位置作为这一帧TLD的输出结果，然后用这个结果更新追踪器的起始位置（Detection->Tracking）。

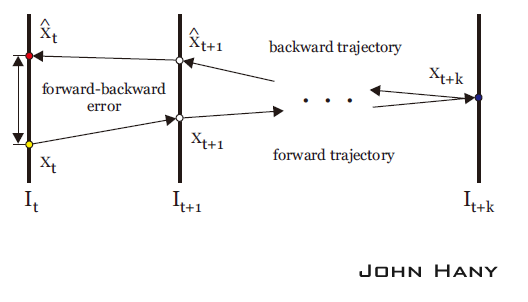
**学习模块**根据追踪器和检测器产生的正负样本，迭代训练分类器，改善检测器的精度（Learning->Detection）。

**追踪模块**

        TLD使用作者自己提出的**Median-Flow**追踪算法。

        作者假设一个“好”的追踪算法应该具有正反向连续性（forward-backward consistency），即无论是按照时间上的正序追踪还是反序追踪，产生的轨迹应该是一样的。作者根据这个性质规定了任意一个追踪器的**FB误差**（forward-backward error）：从时间t的初始位置x(t)开始追踪产生时间t+p的位置x(t+p)，再从位置x(t+p)反向追踪产生时间t的预测位置x`(t)，初始位置和预测位置之间的欧氏距离就作为追踪器在t时间的FB误差。

[](http://johnhany.net/2014/05/tld-the-theory/#image-569)

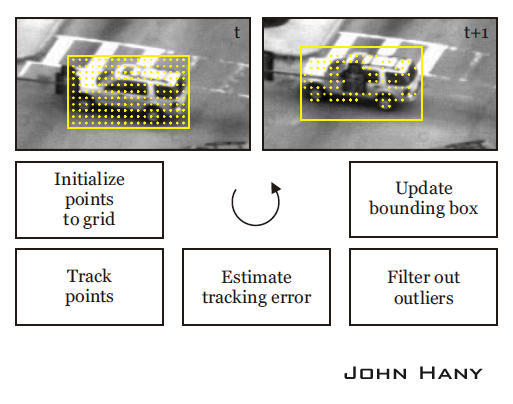
[[](http://johnhany.net/2014/05/tld-the-theory/#imageclose-569)](http://johnhany.net/2014/05/tld-the-theory/" \l "imageclose-569)

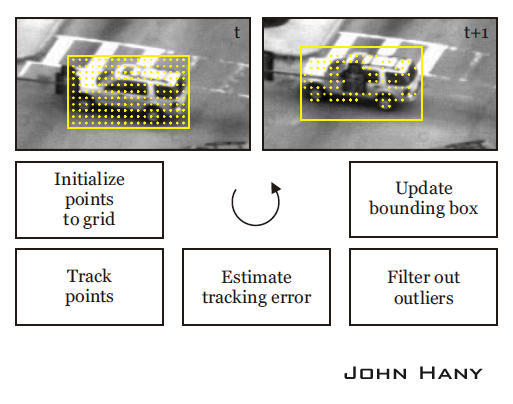
        Median-Flow追踪算法采用的是**Lucas-Kanade**追踪器，也就是常说的光流法追踪器。这个追踪器的原理就不在这里解释了。只需要知道给定若干追踪点，追踪器会根据像素的运动情况确定这些追踪点在下一帧的位置。

**追踪点的选择**

        作者给出了一种依据FB误差绘制误差图（Error Map）筛选最佳追踪点的方法，但并不适用于实时追踪任务，就不详细介绍了。这里只介绍在TLD中确定追踪点的方法。

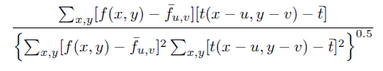
        首先在上一帧t的物体包围框里均匀地产生一些点，然后用Lucas-Kanade追踪器正向追踪这些点到t+1帧，再反向追踪到t帧，计算FB误差，筛选出FB误差最小的一半点作为最佳追踪点。最后根据这些点的坐标变化和距离的变化计算t+1帧包围框的位置和大小（平移的尺度取中值，缩放的尺度取中值。取中值的光流法，估计这也是名称Median-Flow的由来吧）。

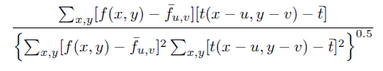
[](http://johnhany.net/2014/05/tld-the-theory/#image-570)

[[](http://johnhany.net/2014/05/tld-the-theory/#imageclose-570)](http://johnhany.net/2014/05/tld-the-theory/" \l "imageclose-570)

        还可以用**NCC**（Normalized Cross Correlation，归一化互相关）和**SSD**（Sum-of-Squared Differences，差值平方和）作为筛选追踪点的衡量标准。作者的代码中是把FB误差和NCC结合起来的，所以筛选出的追踪点比原来一半还要少。

        NCC：

[](http://johnhany.net/2014/05/tld-the-theory/#image-571)

[[](http://johnhany.net/2014/05/tld-the-theory/#imageclose-571)](http://johnhany.net/2014/05/tld-the-theory/" \l "imageclose-571)

**学习模块**

        TLD使用的机器学习方法是作者提出的**P-N学习**（P-N Learning）。P-N学习是一种半监督的机器学习算法，它针对检测器对样本分类时产生的两种错误提供了两种“专家”进行纠正：

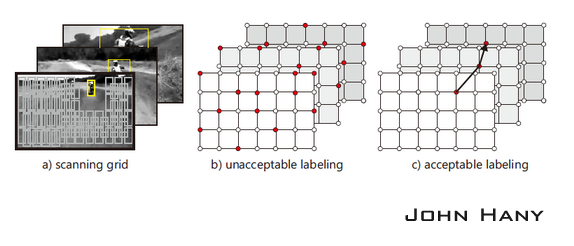
**P专家**（P-expert）：检出漏检（false negative，正样本误分为负样本）的正样本；

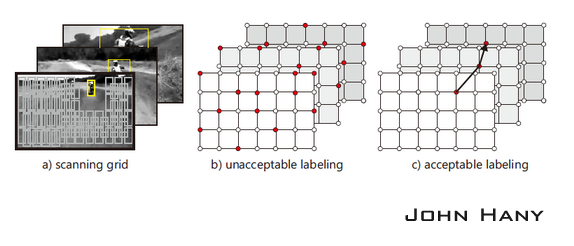
**N专家**（N-expert）：改正误检（false positive，负样本误分为正样本）的正样本。

**样本的产生**

        用不同尺寸的**扫描窗**（scanning grid）对图像进行逐行扫描，每在一个位置就形成一个**包围框**（bounding box），包围框所确定的图像区域称为一个**图像元**（patch），图像元进入机器学习的样本集就成为一个**样本**。扫描产生的样本是未标签样本，需要用分类器来分类，确定它的标签。

        如果算法已经确定物体在t+1帧的位置（实际上是确定了相应包围框的位置），从检测器产生的包围框中筛选出10个与它距离最近的包围框（两个包围框的交的面积除以并的面积大于0.7），对每个包围框做微小的仿射变换（平移10%、缩放10%、旋转10°以内），产生20个图像元，这样就产生200个正样本。再选出若干距离较远的包围框（交的面积除以并的面积小于0.2），产生负样本。这样产生的样本是已标签的样本，把这些样本放入训练集，用于更新分类器的参数。下图中的a图展示的是扫描窗的例子。

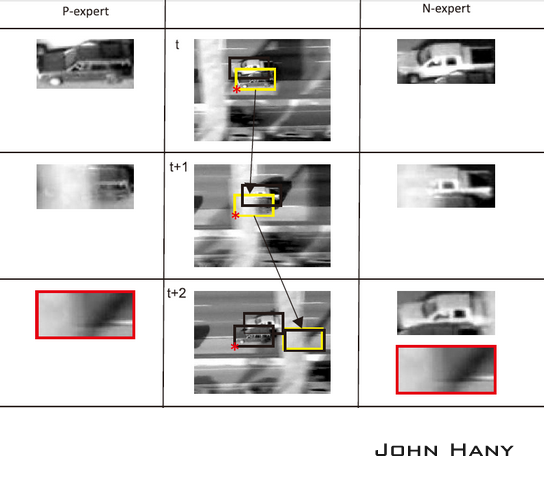
[](http://johnhany.net/2014/05/tld-the-theory/#image-572)

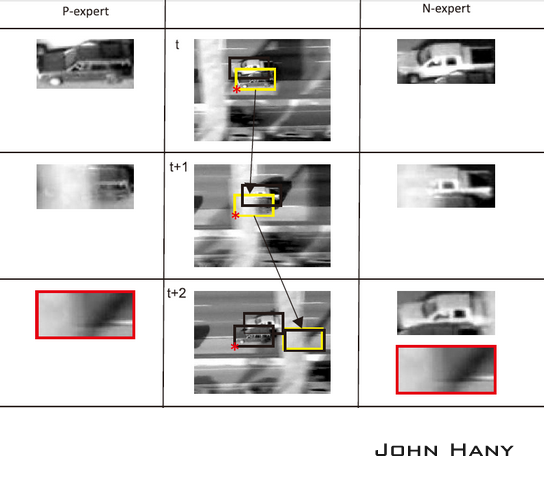
[[](http://johnhany.net/2014/05/tld-the-theory/#imageclose-572)](http://johnhany.net/2014/05/tld-the-theory/" \l "imageclose-572)

        作者认为，算法的结果应该具有“结构性”：每一帧图像内物体最多只出现在一个位置；相邻帧间物体的运动是连续的，连续帧的位置可以构成一条较平滑的轨迹。比如像上图c图那样每帧只有一个正的结果，而且连续帧的结果构成了一条平滑的轨迹，而不是像b图那样有很多结果而且无法形成轨迹。还应该注意在整个追踪过程中，轨迹可能是分段的，因为物体有可能中途消失，之后再度出现。

**P专家**的作用是寻找数据在时间上的结构性，它利用追踪器的结果预测物体在t+1帧的位置。如果这个位置（包围框）被检测器分类为负，P专家就把这个位置改为正。也就是说P专家要保证物体在连续帧上出现的位置可以构成连续的轨迹；

**N专家**的作用是寻找数据在空间上的结构性，它把检测器产生的和P专家产生的所有正样本进行比较，选择出一个最可信的位置，保证物体最多只出现在一个位置上，把这个位置作为TLD算法的追踪结果。同时这个位置也用来重新初始化追踪器。

[](http://johnhany.net/2014/05/tld-the-theory/#image-573)

[[](http://johnhany.net/2014/05/tld-the-theory/#imageclose-573)](http://johnhany.net/2014/05/tld-the-theory/" \l "imageclose-573)

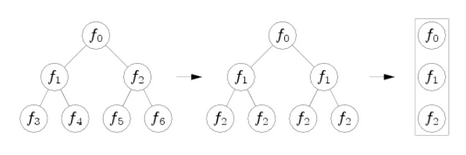
        比如在这个例子中，目标车辆是下面的深色车，每一帧中黑色框是检测器检测到的正样本，黄色框是追踪器产生的正样本，红星标记的是每一帧最后的追踪结果。在第t帧，检测器没有发现深色车，但P专家根据追踪器的结果认为深色车也是正样本，N专家经过比较，认为深色车的样本更可信，所以把浅色车输出为负样本。第t+1帧的过程与之类似。第t+2帧时，P专家产生了错误的结果，但经过N专家的比较，又把这个结果排除了，算法仍然可以追踪到正确的车辆。

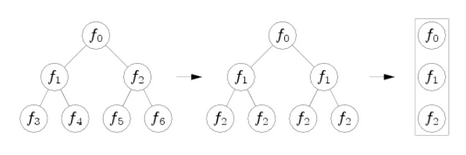
**检测模块**

        检测模块使用一个**级联分类器**，对从包围框获得的样本进行分类。级联分类器包含三个级别：

**图像元方差分类器**（Patch Variance Classifier）。计算图像元像素灰度值的方差，把方差小于原始图像元方差一半的样本标记为负。论文提到在这一步可以排除掉一半以上的样本。

**集成分类器**（Ensemble Classifier）。实际上是一个**随机蕨分类器**（Random Ferns Classifier），类似于随机森林（Random Forest），区别在于随机森林的树中每层节点判断准则不同，而随机蕨的“蕨”中每层只有一种判断准则。

[](http://johnhany.net/2014/05/tld-the-theory/#image-574)

[[](http://johnhany.net/2014/05/tld-the-theory/#imageclose-574)](http://johnhany.net/2014/05/tld-the-theory/" \l "imageclose-574)

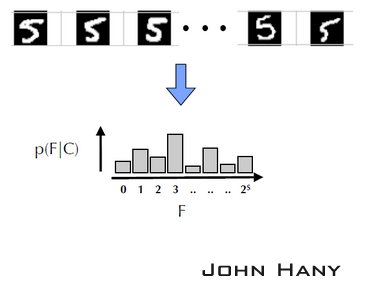
        如上图所示，把左面的树每层节点改成相同的判断条件，就变成了右面的蕨。所以蕨也不再是树状结构，而是线性结构。随机蕨分类器根据样本的特征值判断其分类。从图像元中任意选取两点A和B，比较这两点的亮度值，若A的亮度大于B，则特征值为1，否则为0。每选取一对新位置，就是一个新的特征值。蕨的每个节点就是对一对像素点进行比较。

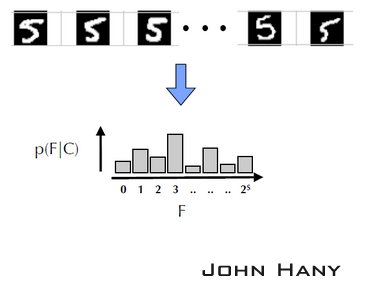
        比如取5对点，红色为A，蓝色为B，样本图像经过含有5个节点的蕨，每个节点的结果按顺序排列起来，得到长度为5的二进制序列01011，转化成十进制数字11。这个11就是该样本经过这个蕨得到的结果。

[](http://johnhany.net/2014/05/tld-the-theory/#image-575)

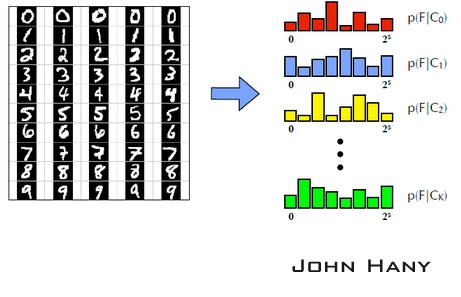
[[](http://johnhany.net/2014/05/tld-the-theory/#imageclose-575)](http://johnhany.net/2014/05/tld-the-theory/" \l "imageclose-575)

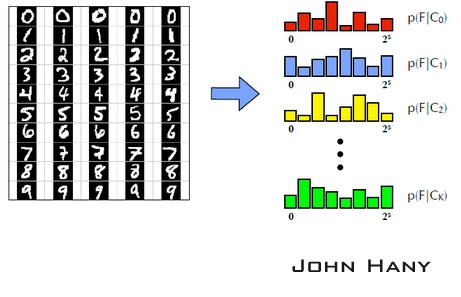
        同一类的很多个样本经过同一个蕨，得到了该类结果的分布直方图。高度代表类的先验概率p(F|C)，F代表蕨的结果（如果蕨有s个节点，则共有1+2^s种结果）。

[](http://johnhany.net/2014/05/tld-the-theory/#image-576)

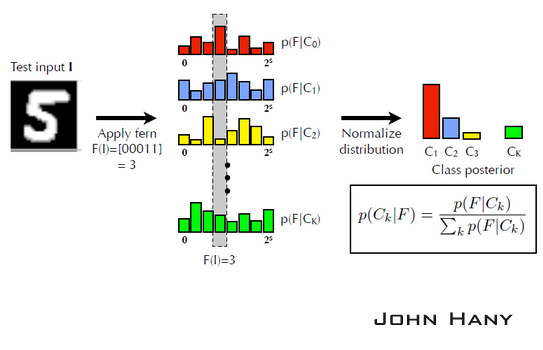
[[](http://johnhany.net/2014/05/tld-the-theory/#imageclose-576)](http://johnhany.net/2014/05/tld-the-theory/" \l "imageclose-576)

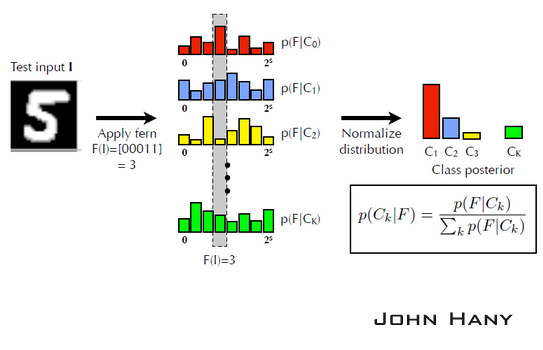
        不同类的样本经过同一个蕨，得到不同的先验概率分布。

[](http://johnhany.net/2014/05/tld-the-theory/#image-577)

[[](http://johnhany.net/2014/05/tld-the-theory/#imageclose-577)](http://johnhany.net/2014/05/tld-the-theory/" \l "imageclose-577)

        以上过程可以视为对分类器的训练。当有新的未标签样本加入时，假设它经过这个蕨的结果为00011（即3），然后从已知的分布中寻找后验概率最大的一个。由于样本集固定时，右下角公式的分母是相同的，所以只要找在F=3时高度最大的那一类，就是新样本的分类。

[](http://johnhany.net/2014/05/tld-the-theory/#image-578)

[[](http://johnhany.net/2014/05/tld-the-theory/#imageclose-578)](http://johnhany.net/2014/05/tld-the-theory/" \l "imageclose-578)

        只用一个蕨进行分类会有较大的偶然性。另取5个新的特征值就可以构成新的蕨。用很多个蕨对同一样本分类，投票数最大的类就作为新样本的分类，这样在很大程度上提高了分类器的准确度。

**最近邻分类器**（Nearest Neighbor Classifier）。计算新样本的相对相似度，如大于0.6，则认为是正样本。相似度规定如下：

        图像元pi和pj的相似度，公式里的N是规范化的相关系数，所以S的取值范围就在[0,1]之间，

[similarity-1](http://johnhany.net/2014/05/tld-the-theory/#image-579)

[[image-579](http://johnhany.net/2014/05/tld-the-theory/#imageclose-579)](http://johnhany.net/2014/05/tld-the-theory/" \l "imageclose-579)

        正最近邻相似度，

[similarity-2](http://johnhany.net/2014/05/tld-the-theory/#image-580)

[[image-580](http://johnhany.net/2014/05/tld-the-theory/#imageclose-580)](http://johnhany.net/2014/05/tld-the-theory/" \l "imageclose-580)

        负最近邻相似度，

[similarity-3](http://johnhany.net/2014/05/tld-the-theory/#image-581)

[[image-581](http://johnhany.net/2014/05/tld-the-theory/#imageclose-581)](http://johnhany.net/2014/05/tld-the-theory/" \l "imageclose-581)

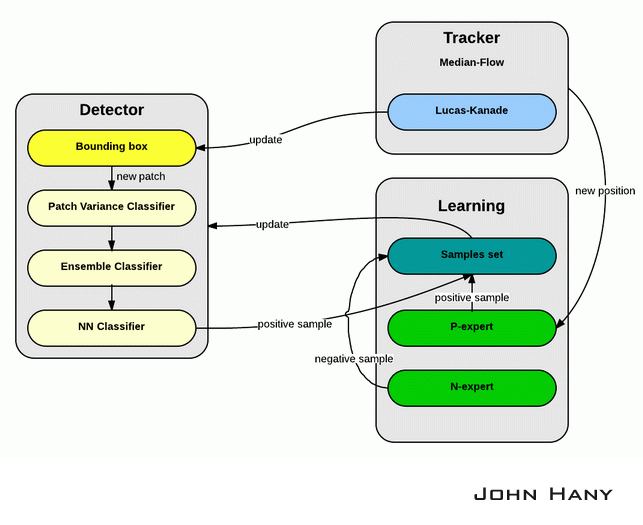
        相对相似度，取值范围在[0,1]之间，值越大代表相似度越高，

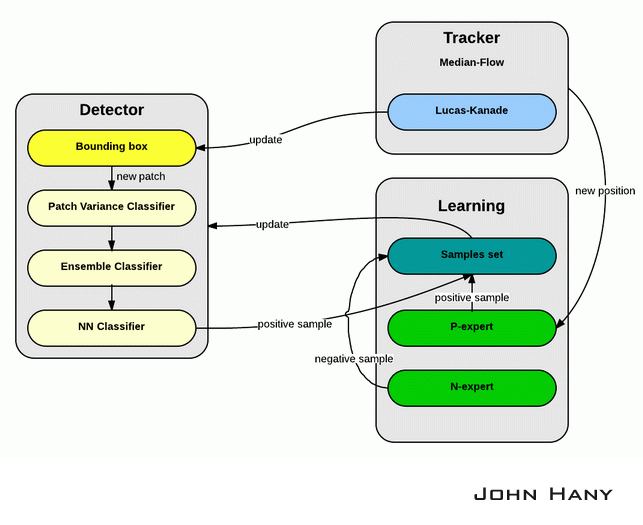
[similarity-4](http://johnhany.net/2014/05/tld-the-theory/#image-582)

[[image-582](http://johnhany.net/2014/05/tld-the-theory/#imageclose-582)](http://johnhany.net/2014/05/tld-the-theory/" \l "imageclose-582)

        所以，检测器是追踪器的监督者，因为检测器要改正追踪器的错误；而追踪器是训练检测器时的监督者，因为要用追踪器的结果对检测器的分类结果进行监督。用另一段程序对训练过程进行监督，而不是由人来监督，这也是称P-N学习为“半监督”机器学习的原因。

        TLD的工作流程如下图所示。首先，检测器由一系列包围框产生样本，经过级联分类器产生正样本，放入样本集；然后使用追踪器估计出物体的新位置，P专家根据这个位置又产生正样本，N专家从这些正样本里选出一个最可信的，同时把其他正样本标记为负；最后用正样本更新检测器的分类器参数，并确定下一帧物体包围框的位置。

[](http://johnhany.net/2014/05/tld-the-theory/#image-583)

[[](http://johnhany.net/2014/05/tld-the-theory/#imageclose-583)](http://johnhany.net/2014/05/tld-the-theory/" \l "imageclose-583)

参考文献：

1. [Tracking-Learning-Detection](http://epubs.surrey.ac.uk/713800/1/Kalal-PAMI-2011(1).pdf), Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2010

2. [Forward-Backward Error: Automatic Detection of Tracking Failures](http://eprints.pascal-network.org/archive/00006952/01/icpr2010.pdf?origin=publication_detail), Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas, International Conference on Pattern Recognition, 2010

3. [Tracking-Learning-Detection Documentation](https://github.com/Ninjakannon/BPTLD)

4. [Decision Trees, Random Forests and Random Ferns](http://cvpr11.cecs.anu.edu.au/files/RandomForests.pdf)

5. [Random Forests and Ferns](http://vision.cse.psu.edu/seminars/talks/2009/random_tff/ForestsAndFernsTalk.pdf)

# TLD算法流程说明

TLD(Tracking Learning Detector)，包括tracking、modeling、detection，其中tracking工作时基于Lucas-Kanade光流法的。modeling学习的过程可以正负反馈，得到较好的学习结果，对于他的学习过程，作者在另一篇文章中又详细介绍了P-N learning这种学习算法。detection的部分用的是多个分类器的级联。对于特征的选择，他提出了一种基于LBP特征的2bit BP特征。

以下解读完全基于arthurv的程序，中间参考了BeyondEgo和zouxy09的博客，在此也表示感谢，由于个人能力有限，难免有不足纰漏之处，欢迎指正，转载请注明出处，谢谢！

整个算法是按照以下流程进行：

一、    读取参数

通过myParam.yml获取参数，通过文件或者用户鼠标选中初始跟踪区域Bounding Box。

二、    用第一帧图像last\_gray、Bounding Box进行初始化工作，调用tld.init(last\_gray,box,bb\_file)，其中bb\_file用于存储后面的跟踪结果。init()具体分为如下几个部分：

1.      buildGrid(frame,box)

这个函数输入是当前帧frame和初始跟踪区域box，主要功能是获取当前帧中的所有扫描窗口，这些窗口有21个尺度，缩放系数是1.2，也即以1为中间尺度，进行10次1.2倍的缩小和10次1.2倍的放大。在每个尺度下，扫描窗口的步长为宽高的10%(SHIFT=0.1)，从而得到所有的扫描窗口存放在容器grid中，这个grid的每个元素包含6个属性：当前扫描窗口左上角的x坐标、y坐标、宽度w、高度h、与初始跟踪区域box的重叠率overlap、当前扫描窗口所在的尺度sidx。其中重叠率的定义是：两个窗口的交集/两个窗口的并集。与此同时，如果当前窗口的宽度和高度不小于最小窗口的阈值min\_win(固定值15，从myParam.yml中获取)，就将这个尺寸放到scales容器中，由于有min\_win的限制，所以某些特别消的尺度下的扫描窗口尺寸就不会放到scales中，故该容器长度小于等于21。

2.      为各种变量或容器分配存储空间，包括积分图iisum、平方积分图iisqsum、存放good\_boxes和存放bad\_boxes的容器，等等。

3.      getOverlappingBoxes(box,num\_closest\_init)

这个函数输入是初始跟踪窗口box，以及good\_boxes中要放入的扫描窗口的数量closest\_init(初始值10是从myParam.yml中获取)，主要功能是：

A.      找出与初始跟踪窗口box重叠率最高的扫描窗口best\_box

B.      找出与初始跟踪窗口box重叠率超过0.6的，将其当成较好的扫描窗口，可以用于后面产生正样本，放到good\_boxes中

C.      找出与初始跟踪窗口box重叠率小于0.2的，将其当成较差的扫描窗口，可以用于后面产生负样本，放到bad\_boxes中

D.      调用getBBHull()函数，获取good\_boxes中所有窗口并集的外接矩形

4.      classifier.prepare(scales)

这个函数输入是buildGrid函数中得到的scales容器，主要功能是进行分类器的准备。TLD中的分类器由三个部分组成：方差分类器(用于结合积分平方图以剔除bad\_boxes中方差较小的负样本)、Fern分类器、最近邻NN分类器。这三个分类器是级联的，也就是说每个扫描窗口必须都通过这些分类器，才能认为是含有前景目标的，才可能是当前帧的跟踪结果。这里的准备主要是针对Fern分类器进行的，所以先说说Fern分类器，以便于说明这个函数做了哪些准备。

Fern翻译是“蕨类”，但从自己的知识构成看，更愿意理解为森林，这个森林有nstructs(值为10)棵树对应nstructs个基本的分类器，每个基本的分类器有structSize(值为13)个节点，每个节点的作用就是根据某种判定规则指定这个节点要往左子树还是往右子树进行搜索，最终搜索到leaf。因此这棵树可以看成深度也是structSize层，每层只有一个节点，这样每棵树其实就是一条从root到leaf的一条路径而没有其它分支。后来再搜索关于蕨类植物的照片，发现用这个词确实更为准确，它就是从根茎部长出多个叶柄(对应多棵树)，每个叶柄上有多个关节(对应节点)，每个关节上左右生出羽片(没有对应，可以忽略)，二每个叶柄上也有一个孢子囊群(对应叶子)。如果还要更形象一点，可以类比为10爪鱼，这个10爪鱼有10个爪，每个爪上有13个关节。

[](http://photo.blog.sina.com.cn/showpic.html#blogid=70a384770101m8eg&url=http://s8.sinaimg.cn/orignal/70a38477g7cf20e87a057)             [](http://photo.blog.sina.com.cn/showpic.html#blogid=70a384770101m8eg&url=http://s13.sinaimg.cn/orignal/70a38477ge1749287274c)

每个节点在进行判断的时候，所依据的规则是：在当前输入的图像片patch中随机去两个点(x1,y1)和(x2,y2)，计算这两点的像素值f1和f2，若f1>f2则返回1，也即往右子树的路径走，反之返回0，往左子树的路径走。而这里对Fern分类器的准备工作就是为features[s][i]申请空间，这里的s表示scales中的尺度索引，取值范围是[0,scales.size()-1]之间的整数，i表示第i个节点，取值范围是[0,129]，这里的129是10\*13-1得到的，对应这个森林中所有树的所有节点。从而features[s][i]表示当前尺度s下第i个节点存储的两个随机点的坐标，其坐标是由当前尺度下的宽高度乘以一个(0,1)之间的随机数得到的，在用每个节点确定路径的时候，就是用此节点的两个随机坐标在输入图像片patch中的像素值进行比较来得到0或1，每棵树的13个节点就得到一个长度为13的二进制编码x，这样的二进制编码取值有2.^13种可能，而每一个x又对应一个后验概率post=pNum/nNum，其中pNum和nNum分别对应正负样本的个数。由于森林(Fern)中有10棵树，从而可以得到10个后验概率，将10个后验概率平均后若结果大于阈值thr\_nn(初始经验值是0.65，从myParam.yml中获取，后面会更新)，则认为该图像片patch含有跟踪目标。由于x有2.^13种可能，所以后验概率post、正样本数目pNum和负样本数目nNum的容器规模也是2.^13，在本函数的最后也就是为其申请空间。

5.      generatePositiveData(frame,num\_warps\_init)

这个函数的输入是当前帧frame、要进行仿射变换的次数num\_warps\_init(初始值20是从myParam.yml中获取)，主要功能是将good\_boxes中的10个扫描窗口进行num\_warps\_init次仿射变换，这样共得到10\*20个窗口，做为正样本数据。可以分为以下几个步骤：

5.1     调用getPattern(frame(best\_box),pEx,mean,stdev)，对于frame的best\_box区域，将其缩放为patch\_size\*patch\_size(15\*15)大小然后计算均值与方差，将每个元素的像素值减去均值使得得到的patern的均值为0。

5.2     调用GaussianBlur(frame,img,Size(9,9),1.5)，对整个frame进行高斯平滑以得到img，高斯模板大小为9\*9，X方向的方差为1.5，同时利用第3步得到的bbhull获取img的该区域。进行仿射变换是用cv::PatchGenerator做为生成器，调用其关于“()”符号的重载运算函数来进行。注意这里感觉并没有用到防身变换后的结果来进行处理，所以可能是代码有误。代码中是用generator(frame,pt,warped,bbhull.size(),rng)，其中frame是当前的帧，pt是bbhull窗口的中心坐标，warped是进行仿射变换后的结果，bbhull.size()是bbhull的宽高度，rng是一个随机数。这个函数是根据bbhull.size()的尺寸和rng生成一个仿射矩阵，对frame进行仿射变换得到结果放到warped中，也即变换前尺寸是原始frame的尺寸，变换后就是bbhull的尺寸了。具体可以参见planardetect.cpp中关于“()”运算符的重载函数。然而其后面并没有对warped进行任何处理，所以感觉有错误。后来查看beyondEgo的博客(<http://quandb2007.blog.163.com/blog/static/4187887520135873314523/>)看到里面的说明也印证了有别人和自己一样的想法。里面提到了两种修改方法，现罗列如下：

法一：将generator(frame,pt,warped,bbhull.size(),rng)　改成　generator(img,pt,img,frame.size(),rng)

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));

                                                         }

                                            }

}

对于法一，是把进行仿射变换的“源”从高斯平滑前的frame换成高斯平滑后的img，其结果也存放到img中，也即仿射变换后的尺寸仍然是img.size()，对于法一还好理解。但是对于法二，其仿射变换变成了generator(img,pt,warped,bbhull.size(),rng)，后面却用了个rect来限定这个区域，而rect左上角坐标的设置看起来挺费解，感觉就是将这个区域由原来的位置进行一点小移动并没有改变区域的形状。这里的修改还是有待验证和深入理解。

总之，这里是对good\_boxes中每个扫描窗口进行20次仿射变换后得到200个patch，然后把patch放到getFeature(patch,grid[idx].sidx,fern)函数中，按照第4点中所述的规则得到长度为13的01二进制编码放到fern中，由于fern中有10棵树，故其规模为10，每个元素存放一个int型值，这个int型值写成二进制形式就是对应该树的长度为13的01编码。之后把这些特征放到正样本数据集pX中，其中makepair(fern,1)中第二个参数”1”表示正样本的类别标签，进而pX中会有200个正样本。

6.      meanStdDev(frame1(best\_box),mean,stdev)

统计best\_box的均值和标准差，取var为“标准差平方的一半”做为方差分类器的阈值

7.      integral(frame1,iisum,iisqsum)

计算积分图和平方积分图，并调用getVar(best\_box, iisum, iisqsum)利用积分图和平方积分图进行快速计算以得到best\_box中的方差，取方差的一半做为检测方差vr。

8.      generateNegativeData(frame1)

由于在跟踪时需要注意到跟踪的目标是否发生远离或靠近镜头，以及旋转和位移的变化所以加入了仿射变换，但是对于负样本而言，它本身就是负样本，进行仿射变换没有什么意义，所以无需进行。由于在3步中重叠率小于0.2的都放到bad\_boxes中，所以其规模很大，这里就先把bad\_boxes中的元素顺序打乱，之后把方差大于var\*0.5的bad\_boxes放到pEx中作为负样本，而把方差较小的进行剔除。和第5步一样，也调用getFeature()获取负样本数据并放到nX中。另外，它还对打乱顺序的bad\_boxes取了前bad\_patches(固定值为100，从myParam.yml中读取)个，通过getPattern()将获取的pattern放到nEx中，作为近邻数据集的训练集用于后面对近邻分类器的训练。

generatePositiveData和generateNegativeData的区别在于：前者进行仿射变换，后者没有；前者无需打乱good\_boxes，而后者打乱了bad\_boxes，目的是为了随机获取前bad\_patches个负样本作为后面近邻数据集的负样本训练集（关于近邻数据集的正样本训练集只有一个数据就是best\_box）

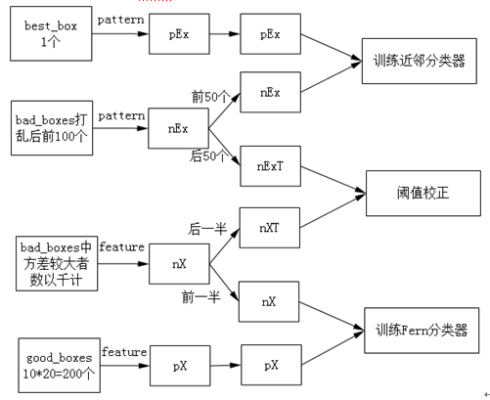
9.      将负样本集nX一分为二成为训练集nX和测试集nXT，将负样本近邻数据集nEx也一分为二成为近邻数据训练集nEx和近邻数据测试集nExT，而正样本集pX和正样本近邻数据集pEx无需划分。

10.     将上面所得的数据集进行如下划分：

10.1    将正样本集pX(good\_boxes的区域仿射变换后的200个结果)和负样本集nX(bad\_boxes中方差较大的区域选取一半出来)顺序打乱放到ferns\_data[]中，用于训练Fern分类器，注意这里对于每个box所存的数据是10棵树分别对这个box的特征：也即10个长度为13的二进制编码。

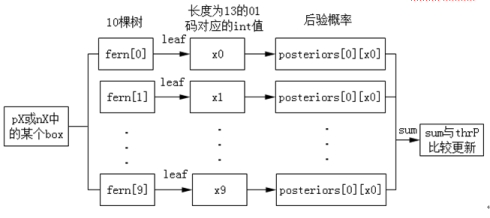
10.2    将正样本近邻数据集pEx(其实只有一个数据，就是best\_box所得到的pattern)和负样本近邻数据集nEx(bad\_boxes打乱顺序后前bad\_patches/2=50个数据所得到的pattern)放到nn\_data[]中，用于训练最近邻分类器。

10.3    负样本集的另一个nXT和负样本近邻数据集的另一半nExT组合起来，作为测试集，用于评价并修改得到最好的分类器阈值。

[](http://photo.blog.sina.com.cn/showpic.html#blogid=70a384770101m8eg&url=http://s2.sinaimg.cn/orignal/70a38477ge17494c7d471)

11.     classifier.trainF(ferns\_data,2)

这个函数输入是正负样本集pX和nX所存储的ferns\_data[]，第二个参数2表示bootstrap=2(但在函数中并没有看出其作用)。主要功能是：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，找出该样本box中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，,0.6这个经验值是Fern分类器的阈值，，初始化时从myParam.yml中读取，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使pNum=pNum+1，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，就添加到负样本库使nNum=nNum+1。update函数有三个参数，第一个参数是该box对应的10棵树fern[]，第二个参数要进行更新的是正样本库还是负样本库，1表示更新正样本库的数目，0表示更新负样本库的数目，第三个参数表示要更新的数目，在整个程序中所有的调用该值都是取1，也即每次都是对样本库数目增加1(**为何另一边的不用相应地减小1？**)，这也跟上面提到的分错的样本放到对应的库是同样的意思，因为每次只能判断一个样本是否有分错，所以更新的数目也只能是1。而在更新数目的同时，也更新了后验概率值，按照post=pNum/nNum的式子来更新

[](http://photo.blog.sina.com.cn/showpic.html#blogid=70a384770101m8eg&url=http://s7.sinaimg.cn/orignal/70a38477ge17495c81d46)

12.     classifier.trainNN(nn\_data)

这个函数的输入是正样本近邻数据集pEx和负样本近邻数据集nEx所存储的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。

12.1    NNConf()函数中对于输入的图像片patch，先遍历在线模型中的正样本近邻数据集pEx中的box(第一次其实就是best\_box，后面会在线更新)，调用matchTemplate()计算匹配度ncc，再由ncc得到相似度nccP，并找出ncc中的最大者maxP；同样的方式也遍历在线模型中的负样本近邻数据集nEx中的box来找出maxN。

12.2    接下来会计算1-maxP和1-maxN，个人理解为：正样本不相似的最小程度、负样本不相似的最小程度。然后就计算相关相似度rsconf=(1-maxN)/[(1-maxP)+(1-maxN)]，也即正负样本不相似的最小情况下，负样本不相似所占比例就定义为相关相似度，如果负样本不相似所占比重越大，那么该patchpatch与负样本不相似的可能性越大，从而相关相似度越高(好像有点拗口，个人是从对偶问题的角度理解的)。

12.3    关于保守相似度csconf是这样得到的，在pEx的前半部分数据中，如果有对maxP进行更新，那么把此时的maxP放到csmaxP中，也即csmaxP记录正样本近邻数据集pEx的前半部分数据中与输入图像片的最大相似度，然后csconf=(1-maxN)/[(1-maxN)+(1-csmaxP)]，由于csmaxP不超过maxP，所以csconf不超过rsconf，也即在认为当前输入图像片patch与正样本近邻数据集数据相似的问题上，对同个patch，rsconf的度量更为“保守”。在第一个进行训练的时候，是否保守没有多大意义，所以在init()中第一次进行trainNN()时，会将rsconf所得到的值丢弃。

12.4    isin中存放三个int型值，初始化全为-1。第一个如果取值为1，则表示NNConf()在计算输入图像片patch与在线模型pEx中的box时发现在线模型中有一个与其相似度超过阈值ncc\_thesame (固定值0.95，从myParam.yml中读取)，此时会把这个patch也放到在线模型的pEx中，所以第一个取值为1就表示已经把当前输入图像片patch放到pEx中。第二个的取值依赖于第1个的取值，如果第一个取值为-1，那么第二个的取值就是-1，如果第一个的取值是1，那么第二个的取值就是在遍历在线模型时找到的第一个与输入图像片patch相似度超过ncc\_the same的box的索引。第三个意义与第一个接近，不同的地方只在于第一个是对应在线模型的正样本近邻数据集pEx，第三个是对应在线模型的负样本近邻数据集nEx。

13.     classifier.evaluateTh(nXT,nExT)

这个函数的输入是负样本数据集的后半部分nXT(存放feature)和负样本近邻数据集(存放pattern)的后半部分nExT，主要功能是将这两个作为测试集用来校正阈值thr\_fern、thr\_nn、thr\_nn\_valid。

13.1    更新thr\_fern：对nXT中的每个样本，用measure\_forest()函数找出10棵树对这个box的01码，由01码找到对应的后验概率，将10个后验概率累加后求平均，将平均值与thr\_fern(初始值0.6，从myParam.yml中获取)比较，如果超过thr\_fern则将thr\_fern更新为这个平均值。也即对于nXT中所有的box，10棵树都会对其投票得到一个后验概率，将所有后验概率取平均后，比较所有的box，取后验概率均值最大的那个box所对应的平均值放到thr\_fern中。

13.2    更新thr\_nn：对于负样本近邻数据nExT测试集中的每个样本，用NNConf()函数计算它与在线模型pEx和nEx中数据的相似度来得到相关相似度conf(与第一次训练NN分类器一样，这里得到的保守相似度也被丢弃了)，如果conf大于阈值thr\_nn(初始值0.65，从myParam.yml中获取)，则更新thr\_nn为conf。

13.3    更新thr\_nn\_valid：如果更新后的thr\_nn大于thr\_nn\_valid(初始值0.7，从myParam.yml中获取)，那么更新thr\_nn\_valid值为thr\_nn。

一、     初始化工作进行完毕之后，开始顺序读取每一帧，也即调用tld.processFrame(last\_gray,current\_gray,pts1,pts2,pbox,status,tl,bb\_file)逐帧处理。last\_gray和current\_gray分别是上一帧和当前帧的灰度图；pts1和pts2是跟踪点的坐标，初始化为空，在函数内部实现赋值和应用；pbox是上一帧跟踪的结果；status是bool值用于表示上一帧是否有跟踪到窗口；tl个人理解t对应track，l对应learn，也即tl是表示是否进行track和learn的bool标记；bb\_file是存放跟踪结果的文件的路径。这个函数依次执行以下四个模块：跟踪模块、检测模块、综合模块、学习模块。

1.      跟踪模块：track(img1, img2, points1, points2)

这个函数的输入是上一帧灰度图img1和当前帧灰度图img2，以及跟踪点的坐标points1和points2。它分以下几个步骤进行：

1.1     bbPoints(points1,lastbox)

这个函数主要功能就是进行点的生成，在上一帧的跟踪区域lastbox中进行均匀采样，得到不超过10\*10=100个点放到points1中(因为采样步长是用ceil进一法得到，所以每行或每列得到的点可能无法达到10个)。

1.2     tracker.trackf2f(img1,img2,points1,points2)

这个函数输入是上一帧灰度图img1和当前帧灰度图img2，bbPoints()生成的点序列points1，输出是points2，主要功能是完成：跟踪、计算FB error和匹配相似度sim，然后剔除 匹配度小于匹配度中值的(sim\_error[i] > median(sim\_error))，再剔除跟踪误差大于误差中值的(FB\_error[i] <= median(FB\_error))， 也即把跟踪结果不好的特征点去掉，剩下的是不到50%的特征点，对应地留在points1和points2中。

跟踪的原理基于Forward-Backward Error的中值流跟踪方法，对于points1中的每个点，使用前向跟踪，即上一帧的点A(由于点A是在lastbox行生成的，所以确实是上一帧的点)在当前帧的跟踪结果为B，然后使用后向跟踪，即当前帧的点B反向跟踪得到上一帧的跟踪点C，这样就产生了前向和后向两条跟踪轨迹，理想的情况应该是两条轨迹重合，即A和C是重合的，所以计算A和C的距离FB\_error，得到一个FB\_error[]数组。之后调用normCrossCorrelation()计算A和B的similarity，这个similarity是以A和B为中心的，分别在上一帧和当前帧截取的10\*10的区域调用matchTemplate()函数计算匹配度，将匹配度值赋给similarity，得到一个similarity[]数组。

接下来调用filterPts(vector& points1,vector& points2)对所得到的点进行过滤，其中points1是所有点A组成的集合，points2是所有点B组成的集合。首先计算similarity[]数组中所有数的中值simmed，对于similarity超过simmed的点进行保留，其余的剔除，这样FB\_error[]数组的规模也相应减小。之后计算这个减小规模后的FB\_error[]数组的中值fbmed，对于FB\_error小于fbmed的点进行保留，其余的剔除。

1.3     bbPredict(points1,points2,lastbox,tbb)

这个函数输入中的point1和points2是前面用光流法跟踪并剔除跟踪效果不好的特征点而剩下的点集，lastbox是上一帧的跟踪结果，tbb是用于记录当前帧的跟踪结果。主要功能是利用剩下的这不到一半的跟踪点作为输入来预测bounding box在当前帧的位置和大小 并放到tbb中。 它先按x维度和y维度分别计算所有点在上一帧和当前帧的跟踪差距，并计算出这些差距的中值dx，dy。接下来计算points1中所有点两两之间的距离d1和points2中所有点两两之间的距离d1，将d2/d1都放到d中，计算所有的d的中值s，这个s就表征了进行光流跟踪之后特征点变化相对于上一帧位置的偏移比例，从而用s去乘以上一帧窗口的宽高度以得到当前帧跟踪结果窗口的宽高度；另外，跟踪窗口的左上角坐标也要更新，坐标的偏移不能只考虑位移的绝对值，还要考虑窗口本身宽高度，也即这个位移相对于窗口本身的比例，所以用0.5\*(s-1)分别去乘以上一帧窗口的宽度和高度，得到偏移量s1和s2，再结合表征了上一帧和当前帧跟踪差距的dx和dy，得到新跟踪结果tbb(**track bounding box**)的左上角坐标(lastbox.x + dx -s1, lastbox.y + dy -s2)。

1.4     对跟踪结果进行判断，如果fbmed超过10(固定经验值)、或者窗口坐标位于图像外，说明跟踪的结果不稳定，将跟踪结果丢弃，置tvalid和tracked为false并进入下一帧，否则继续进行下面的过程。

1.5     估计置信度和有效性，调用getPattern()计算当前帧在跟踪结果区域的pattern，把pattern做为输入调用NNConf计算它与在线模型的保守相似度tconf，如果tconf>thr\_nn\_valid，则置tvalid为true，也即表示当前跟踪结果有效，否则tvalid仍为上一帧的值。

2.      检测模块：detect(img2)

这个函数的输入是当前帧的灰度图，对该图中的所有扫描窗口，依次用上面提到的方差分类器、Fern分类器、最近邻分类器所形成的级联分类器进行检测，只有通过所有分类器检测的扫描窗口才认为含有前景目标而可能是跟踪区域。具体过程如下：

2.1     计算积分图和积分平方图，对img2进行高斯平滑。之后对所有扫描窗口，依次进入以下级联着的分类器

2.2     方差检测分类器，利用积分图调用getVar()计算每个扫描窗口的方差，如果方差大于var阈值（var初始初始是best\_box标准差平方的一半，也即目标patch方差的一半），则认为其含有前景目标，(**注意：虽说标准差是方差的平方根，但是程序中求方差和标准差是用两个不同的函数计算的**)。

2.3     Fern分类器，对通过前面方差检测分类器的扫描窗口，调用getFeatures()计算10棵树对该扫描窗口的编码(长度为13的01序列)，利用该编码结合measure\_forest()得到10棵树对该扫描窗口的10个后验概率累加和的平均值，如果平均值大于Fern分类器的阈值fern\_th(初始从myParam.yml获取是0.65后面会不断更新)，则该扫描窗口也通过Fern分类器的检测，将窗口放到容器中以便后面最近邻分类器的检测。(程序中是10个后验概率累加和conf与10\*fern\_th进行比较，其实一样，就是将除法换成乘法，以加快计算速度，同时conf也可用于下面的排序)

2.4     通过前面两个分类器检测的扫描窗口按照后验概率累加和conf进行降序排列，如果窗口数目超过100个，则取前面100个后验概率较大的。对这些窗口，调用getPattern()获取pattern，然后调用NNConf()计算与在线模型的相关相似度conf1和保守相似度conf2，如果相关相似度conf1大于近邻分类器的阈值nn\_th，则保留该窗口及其对应的保守相似度的值，通过以上三个分类器检测的扫描窗口放到dbb容器中，对应的保守相似度放到dconf容器中。

3.      综合模块

这个模块是对跟踪和检测结果进行综合，按照是否检测到、是否跟踪到的两两组合可以分为四种情况：跟踪到检测到、跟踪到但没检测到、没跟踪到但检测到、没跟踪到没检测到。由于需要用到检测模块中的保守相似度，也即必须使用通过检测模块的扫描窗口的信息，所以只考虑第一和第三种情况。

3.1     如果跟踪到并检测到

对检测到的窗口(通过检测模块的三个分类器后得到的窗口)，调用clusterConf(dbb,dconf,cbb,cconf)进行聚类，其中dbb和dconf是检测模块得到的结果(**dbb->detect bounding box**)，cbb(**cbb->cluster bounding box**)和cconf是该函数的输出。这个函数针对dbb中窗口的数目采取不同策略，如果只有一个窗口，那么不用聚类，直接把dbb和dconf分别赋给cbb和cconf即可；如果有两个窗口，调用bbOverlap()计算两个窗口的重叠率，如果重叠率小于一定的阈值1-space\_thr(固定值为0.5)，则分为两类。如果窗口超过两个，那么调用partition()函数按照重叠率将窗口分为两类，一类是重叠率小于0.5的，另一类是超过0.5的。此处自己不是特别理解，参考BeyondEgo和zouxy09的博客，说到“c = partition(dbb,T,(\*bbcomp))”把一个区间中的元素按照某个条件分成两类，并返回第二类子集的起点。bbcomp这个指针所对应的函数是比较两个窗口重叠率，小于0.5返回假，否则返回真。但是他们又提到partition()函数返回的是第二个子类的起点，那么c到底是类别的数目还是第二个子类的起点呢？（最开始以为可以直接按照重叠率排序后选出重叠率超过.5的即可，何必用partition()那么麻烦，后来仔细想想，如果是按照重叠率排序，那么计算的是这些窗口和哪个窗口的重叠率？其实这里并没有哪个参考窗口，所以计算的重叠率其实是这些窗口两两之间的重叠率，然后将重叠率不超过0.5的归为一类，有点像聚类分析中的类内距离）后来找到partition()函数是在\core\operations.hpp中，里面提到“This function splits the input sequence or set into one or more equivalence classes and returns the vector of labels - 0-based class indexes for each element”，所以这个函数应该是分成多个等价类，而不一定是两个，函数返回的c是等价类的数目，T是每个元素所属的类别标签。比如有ABCDEFG七个窗口，经过调用后ABC是一类，两两之间重叠率小于0.5，DE是一类，FG是一类，DE与ABC的所有窗口两两之间重叠率超过0.5，FG与ABC的所有窗口重叠率超过0.5，而DE和FG之间的窗口两两之间重叠率也超过0.5。在聚类结束之后，把每个类别中的窗口信息进行平均，也即ABC所有窗口的x坐标求和再除以该类别数目3，以其作为聚类中心窗口的x坐标，其它信息包括y坐标、宽度、高度、保守相似度也按如此计算，并保存到cbb和cconf中，也即cbb和cconf的规模是聚类的数目。

接下来对cbb中每个聚类中心进行判断，如果有聚类中心满足如下条件一：“它与tbb(也即跟踪模块中光流跟踪得到的当前帧的跟踪区域)的重叠率小于0.5但是保守相似度却大于tbb的保守相似度，也即找到一个聚类中心离跟踪器的跟踪结果较远(重叠率小于0.5)但是比跟踪器更为可信(保守相似度高于tbb的)”，那么记录满足条件一的聚类中心的数目，如果这个数目是1，也即只有一类满足，那么重新初始化跟踪器，并置lastvalid=false表示上一帧在跟踪器中是无效的（也即用检测器的检测结果对跟踪器的跟踪结果进行修正）。如果没有找到满足条件一的聚类中心，则找出检测到检测结果dbb中所有窗口与跟踪结果tbb重叠率超过0.7的，将其与tbb的信息一起累加再求平均，但是跟踪器tbb的权重较大，虽然tbb只有一个窗口，但是在计算平均值的时候是用10个tbb加上其它dbb中的信息再除以总个数，所以跟踪器tbb权重大(一抵十)。

总之，如果有跟踪到(tbb)并检测到(dbb)，那么对检测到的结果聚类得到cbb(dbb->cbb)，判断聚类中心与tbb的重叠率和可信度，如果两者重叠率低于0.5且cbb可信度更高，那么用cbb修正tbb，也即用检测器修正跟踪器；如果重叠率接近，那么用检测器和跟踪器结果加权平均得到结果bbnext，其中跟踪器的权重更大。

3.2     如果没跟踪到但检测到

此时没有tbb只有dbb，置lastboxfound和lastvalid为false表示上一帧的box无效。同样对dbb进行聚类得到cbb，如果聚类中心只有一个，将聚类中心的信息作为当前帧的处理结果bbnext，如果聚类中心有多个，丢弃检测器的检测结果。

对于综合模块，其实就是用跟踪模块和检测模块的结果进行综合考虑，前提是检测器有检测到结果dbb，对dbb进行聚类得到cbb，然后根据跟踪器的结果tbb的情况来分析：如果tbb存在，看tbb与cbb的重叠率，如果重叠率低但是cbb可信度高，用cbb修正tbb；如果重叠率接近，将dbb和cbb加权平均得到当前帧跟踪结果bbnext，但是跟踪器dbb权重较大；如果tbb不存在，看聚类结果tbb是否只有一个中心，是则以其为当前帧跟踪结果bbnext，如果不只一个中心则将聚类结果丢弃，当前帧没有跟踪到。

4.      学习模块learn(img2)

这个函数的输入是当前帧的灰度图，主要功能是进行学习。首先，用上一帧的跟踪区域lastbox在当前帧上截取，并调用getPattern()得到其pattern和方差，再调用NNConf()计算其与在线模型最近邻数据集的相关相似度conf，如果相关相似度太小、或者方差太小、或者被识别为负样本，则不进行训练，lastvalid置为false表示上一帧失效然后直接return。

如果以上条件都不满足，则更新当前帧扫描窗口与上一帧窗口lastbox的重叠率，调用getOverlappingBoxes(lastbox,num\_closest\_update)更新best\_box、good­\_boxes、bad\_boxes、bbhull，此时bad\_boxes中的规模为num\_closest\_update(值为10，从myParam.yml中获取)，也即bad\_boxes的规模一直不变。然后借助good\_boxes结合generatePositiveData(img,num\_warps\_update)清空并重新生成pX(仍然存储仿射变换后所得的200个样本)和fern中每棵树对pX中200个正样本的编码。之后对bad\_boxes中的样本进行判断，如果10棵树对bad\_boxes中的样本得到的10个后验概率之和超过1，则将其作为Fern分类器fern\_examples中的样本，用于后面对Fern分类器的阈值进行更新。

接下来更新近邻数据集nn\_examples，将检测模块中通过方差分类器和Fern分类器的样本dt与lastbox进行重叠率计算看是否不超过bad\_overlap(固定值0.2从myParam.yml中读取)，不超过则可以将其放入近邻数据集nn\_examples中，用于后面近邻分类器的更新训练和阈值的更新。

              这样，逐帧处理的过程也就结束了，如此每帧处理直到结束。

关于代码注释部分就不贴出来了，大部分是写得只有自己才看得懂的，就不献丑了，可以参考前面提到两个的博客