TLD 项目

By Xian2207, 13689903575, wszhangxian@126.com

Github 关于物体跟踪文献：https://github.com/wuxuedaifu/MVision

1 跟踪器分类

Figure 2.2: Classification of trackers based the representation of the object: a) points, b) geometric shapes, c) contours, d) articulated models, and e) motion field.

翻译：跟踪器分为点，几何形状，轮廓，关节模型，运动场。

知识点：

points are the smallest object that do not change their scale dramatically. Algorithms that represent the object by a point will be called point trackers. Typical methods are frame-to-frame tracking(Optical-Flow), key-point matching, key-point classification or linear prediction.

Geometric shape are bounding boxes or ellipses, etc.

Contours are used to represent non-rigid objects(非刚性可变性的物体，如人手，动物)

Articulated models are used to represent the motion of non-rigid objects consisting of several rigid parts. These models consist of several geometric shapes.

Motion field is a non-parametric(非参数型) representation of the object motion which gives the displacement of every pixel of the object between two frames.

2 generative model 和 discriminative model 在 TLD 中的区别

答： Tracking algorithms represent the object by a model. Generative models represent the appearance of object ignoring the environment where the objects moves; Discriminative models focus on the differences between object and the environment. Both of them are either static which means remain fixed scene during tracking or adaptive which accepts the new information during tracking.即生成式模型强调数据的分布情况，反映数据的相似性；判别式模型强调异类数据的差异性。前者可以得到后者，后者无法得到前者。

3 Drift 和 failure 的区别

答：Drift happens in adaptive trackers in long-term tracking. The phenomenon shows that the errors of the update accumulate over time and the tracker slowly slips away from the object. It is different to failure which is a sudden incorrect estimation of object state. Failure is caused by object appearance changes, full occluded or moves out of camera view. 即漂移是因为错误累积，导致跟踪器一点点远离目标；而失败是因为目标外在突然的改变，如外形，颜色，遮挡等。

4 Generative trackers 生成式跟踪器

答：主要代表是模板匹配，利用模型在后续帧进行区域模板匹配。根据区域内图像匹配求解相似度的理念，衍生出了两个主流方法：gradient-based methods 和 mean-shift。前者典型代

表是光流法，计算的是模型的平移量，缺点是容易 Drift 且不能处理遮挡，如果帧间运动平移过大，方法容易失效。后者典型代表是利用目标的颜色分布。基于一幅新图像，求解反向投影图（即该图上每个像素赋予一个概率值后经过处理得到的二值化图），mean-shift 迭代求解反向投影图中类似于上一帧反向投影图的目标。缺点是基于颜色直方图，一旦颜色发生改变或者目标周围有相似颜色目标，容易丢。

知识点：模板匹配跟踪器的两大缺点：[1]模板自适应更新难，导致漂移。即使动态更新模板，也容易与背景混淆，导致漂移;[2]无法有效处理遮挡。即使原始模板处理成几个部分，但对于复杂背景而言，分割的部分模板容易和背景搞混，导致遮挡处理失败。

5 Static Discriminative Trackers 判别式跟踪器

答：主要代表是机器学习分类器，如 SVM，RF 和 NN。SVM 可以区分背景和目标，但属于线下训练，需要大量数据，区分后用光流法跟踪；RF 和 NN 是在线训练，数据量明显少于 SVM。离线训练较早有 04 年的 Avidan 提出的 Support Vector Tracker 跟踪器，Avidan 07 的 Ensemble Tracking（[1]选择某弱分类器训练已经赋予权值的数据（N 个数据，权值为 1/N）得到预测函数；[2]将分类错的数据给予更高的权值，再次训练直到发现最好的弱分类器，并给予该分类器权值；[3]再选一个弱分类器，重复 1 和 2，直到所有分类器训练结束并赋予权值;[4]强分类器就是弱分类器与权值乘积的组合），区分中目标后，用 meanShift 跟踪. 优点是短期跟踪好，稍微遮挡一两次没事，但长期跟踪不行，因为数据一经训练结束后就抛弃了，目标长时间遮挡就找不回来了。

6 Constrained adaption for discriminative trackers 受限制的自适应判别式跟踪器

答：constrained adaption aims at solving drift reduction and long-lasting full occlusions. 常用三个方法是：（1）半监督学习；（2）multiple-instance learning 多实例学；（3）co-training（协同训练）.

7 Sliding Window and non-maxima-suppression 非极大值抑制

答：Sliding window is a grid of locations at multiple scales. 即不同尺度的网格; non-maxima suppression 即最大值抑制，去掉重合度小的或像素值小的目标框或者点。

8 TLD 特征提取

答：一般是 HOG，haar-like，LBP 特征。TLD 论文指出以下为目标检测的主要特征：

Raw Pixels – 原始像素，优点是简单高效；缺点是维数过高，计算慢。相似性的计算使用 Normalized Cross-Correlation(归一化) or Sum of Square Difference SSD(平方差求和).

Histogram – 直方图分布，例如颜色，灰度值等，像素之间空间信息缺失；

Filter – 比对先前定义的特征值在待检测目标的大小，从而检测出目标。典型代表 Haar-like 特征，利用积分图计算特征值。

Gradients – 利用梯度特征检测图片的相似性。典型代表是 SIFT（尺度不变特征变换），原理是检测出特征点，将该点周围分裂成 4x4 图块，在每个图块中产生 8bin 梯度的直方图。梯度是 128 维特征向量。基本过程是：尺度空间搭建，极值点检测，筛选极值点，特征点方向分配，特征点描述。

Codes – 将像素由真实值转换为离散值，典型代表 LBP。

9 TLD Machine Learning

监督学习

[1] Bootstrapping (自助采样法): it is a strategy that iteratively improves a classifier by training it on increasingly larger and more informative subset (active set) of the entire training set (pool). 方法是: (1)在 pool 中随机产生 active set; (2)training the classifier on the active set; (3) testing the classifier on the pool data;(4)enlarging the active set with misclassified examples from the pool; (5)不断重复(2-4), 直到全部数据分类正确。

注意：此法适用于有监督学习，即训练数据都要有 label.

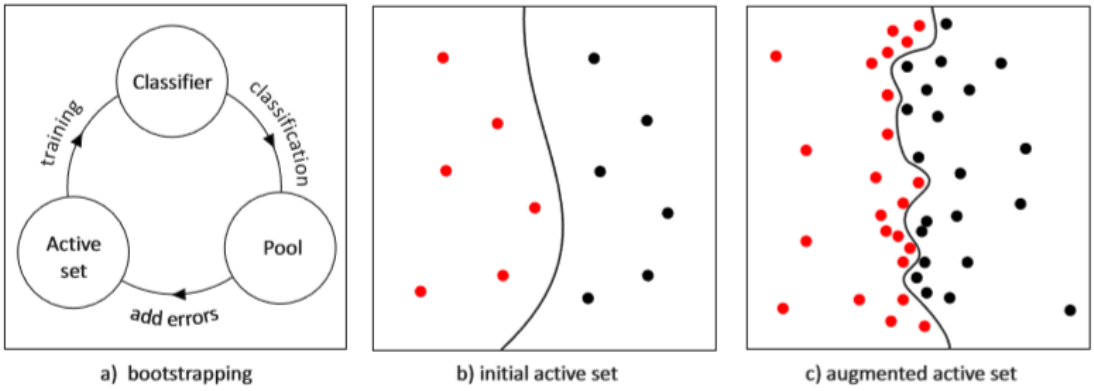


Figure 2.13: Bootstrapping: a) the block diagram, b) an initial training set, c) a training set after several iterations of bootstrapping.

[2] Boosting (提升法): 一种监督学习方法，属于 ensemble methods（集成法）. Consider a labeled training set and a set of arbitrary weak classifiers. Boosting combines these weak classifiers to a strong one which can perform better than any of weak classifiers. 过程是(1) prepare n labeled training data and equal weight say 1/n and several weak classifiers; (2)select the weak classifier that performs best on current distribution of weights; (3) update distribution of weights (often increase weighted values) on misclassified examples; (4)find the weak classifier H that yields the best result based on loss function and give its weight to H, say 预测函数; (5) repeat 2-4 we will find good classifiers H1, H2, …， Hn(预测函数) with different weight; (6)final strong classifier will be polynomial of these good classifiers and their associated weight. Like f = H1*a1 + H2*a2 + H3*a3…Boosting 典型代表就是 Standard AdaBoost 和 Cascaded AdaBoost。

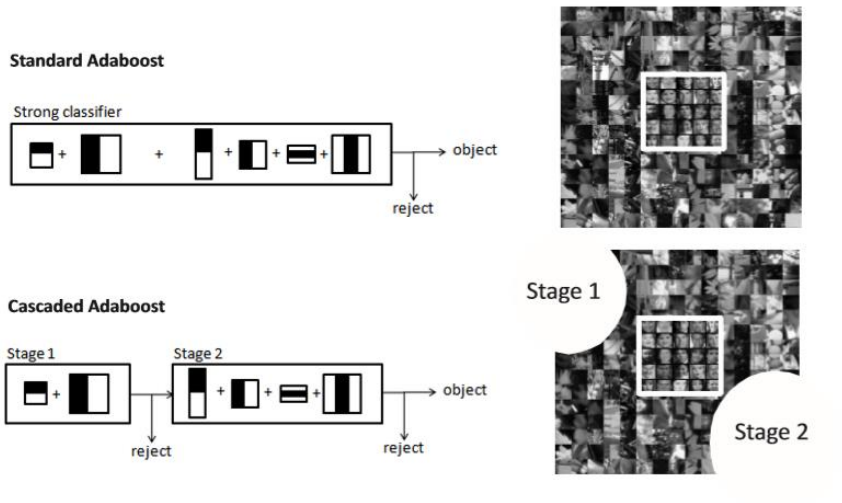
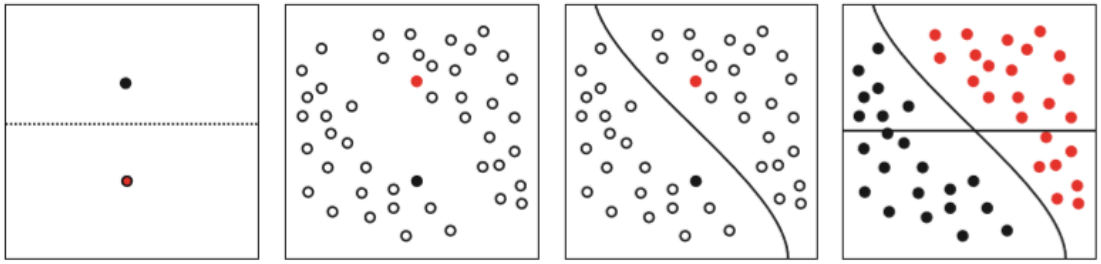


Figure 2.15: Standard AdaBoost versus cascaded AdaBoost: (TOP) standard AdaBoost has to evaluate all weak classifiers (here Haar-like filters), (BOTTOM) cascaded AdaBoost enables to reject negative examples after each stage of the classifier thus reducing the computation demands.

注意：TLD 之所以不用 AdaBoost 和 Cascaded AdaBoost 做采样方法，是因为 sliding window

是 10^5 个不同尺度的网格，约 10 个是包含目标的。故正负样本极不对称，大多数都是负样本，计算开销过大；此外过大的负样本个数导致不太容易分配合适的权值，从而无法有效可靠的实施 boosting learning method.

半监督学习 – 没有标签的样本是否有助于提升分类器？



答：图 a 假设两个有 label 的数据，线性可分；图 b 加入 unlabeled data；图 c 假设如果把两边 unlabeled data 和 labeled data 分成各自所属的一类，那么 unlabeled data 就能产生两个分类器。TLD 用该例子说明 unlabeled data 可以提升分类器的性能。以上即所谓的"cluster assumption". 基于此假设提出三种算法，分别是 Expectation-Maximization(EM), Self-Learning, Co-training 用来给 unlabeled data 打标签或分类。Self-learning process: (1)prepare labeled training data and train initial classifier using this dataset; (2)examples with the highest confidence are added to training set; (3)repeat(1-3) until loss function is satisfied. Self-learning 被用到人眼检测。Co-training process: (1)prepare two separate classifier and train them using labeled data; (2)both are then used to train unlabeled data; (3)没标签样本经过分类器训练后得到正确标签的样本给另一个分类器；同样另一个也将正确分类的无标签样本给第一个分类器，然后再次训练，知道收敛。该方法特别适用于独立模型，如文本分类，生物特征分类，因为样本模型彼此独立且差异大。而 TLD 不采用此法，原因是目标识别跟踪的样本均来自一个模型。TLD 经过自助采样得到的样本彼此是有关联的，有悖于 co-training assumptions.

TLD 初始化

[1] 全局网格生成（滑动窗口）, 75776 个

buildGrid(pregray, box); //pregray 第一帧灰度图，box 是 selectROI 框选的

方法：

A: 设置扫描窗口移动的偏移量 SHIFT；

B: 扫描窗口 21 种尺度 s=[0.16, 6.19];

C: 取 s[0] 时，width = round(box.width*scale)；height=round(box.height*scale); min_bb_width=min(width, height);

D: 边界判断，如果 with, height, min_bb_width 越界，则不分割网格；

E:
```
for (int y = 1; y < pregray.rows - height; y += round(SHIFT*min_bb_side))
    for (int x = 1; x < pregray.cols - width; x += round(SHIFT*min_bb_side))
```
利用上面两层循环，得到 grid 的元素 bbox.x, bbox.y, bbox.width, bbox.height, bb.overlap 及对应的 index；

F: 重复 s[1],s[2]…,s[6.19],重复 C 到 E 步骤，即生成全局网格。

[2]正样本库 pX

getOverlappingBoxes(grid, num_closest);//num_cloest = 10;不涉及原始框和 grid 的重叠度计算

方法：

A:遍历 grid 的 overlap，找到 best_overlap 即和原始 box 重合度最高的；

B:遍历 grid(i)，凡是 grid[i].overlap>0.6，将 i 记录到 good_boxes 里；凡是<0.2 的，记录到

bad_boxes[i]里，自然有部分 grid 介于 0.2 到 0.6，要被舍弃；

good_boxes 有 32 个，bad_boxes74758 个

C: 对 good_boxes 用 nth_element 对 good_boxes 坐在的 grid.overlap 排序，前 10 的保存，后面的一律删除，最好 good_boxes 只保存 num_cloest 即 10 个 grid 的 i；

D: getBBHull(good_boxes)；目的是对 10 个 i 对应的 grid 把位置宽高找到，然后画一个大框将所有相互重合的 good_boxes 框住，即 bbhull.

E: getPositiveData(pregray, num_warp_init); //num_warp_init =20

getPattern(prgray(best_box))；主要是将包含目标的 ROI 缩放成 15*15(patch_size)的 pattern，然后求均值,用 pattern 减去均值，为模板匹配相关系数法做准备；同时，减去均值的 pattern 也就是 p_patch，为 Nearest Neighborhood Clustering 的唯一正样本；

F:高斯模糊，遍历 good_boxe，将 good_boxes[i]对应的 grid 给 getFeature 函数，该函数记录 grid 对应的 fern 值。

G:将 10 行 1 列的 fern 值利用 vector<pair<vector<int>,int>>全部标记成 1，然后 push_back 到 pX 正样本。正样本大小即为 10 行，每行里的元素都是一个 10*1 的 fern 和一个 1；由于 num_warp_init 的存在，将 F 步骤重复 20 次，最终是 pX 是 200 行，每行元素是 10*1 的 fern 和一个 1。显然，这里有重复样本的存在。

[3]负样本 NX

A: randome_shuffle(bad_boxes.begin(), bad_boxes.end());将 74758 个 bad boxes 乱序；

B:遍历负样本库，将 bad_boxes(i)赋给对应的 grid，然后根据 grid[i]裁剪 ROI，计算 ROI 方差。如果 ROI 方差小于（原始 ROI 方差的一半），放弃该 grid；否则传递给 getFeature 函数，经过特征提取得到 fern 和对应的标签 0，最后 push_back 到 NX 负样本库里。最终 nX 有 39398 个，是从 74758 个 bad boxes 里筛选的，可见少了近 1/2;

C: 人为定义 bad_patches = 100; n_patch = vector<Mat>(bad_patches);

```
for (int i = 0; i < bad_patches; i++) {
        idx = bad_boxes[i];
        patch = pregray(grid[idx]);
        getPattern(patch, n_patch[i], dum1, dum2);
}
```

将剩余的 39398 个 bad_boxes 中的 i 找到对应的 grid，然后将该 grid 对应的 ROI 转换为 15*15 pattern,求解均值和方差，再利用 patter 减去均值得到的就是 n_patches，故它的大小为 100 个，为 Nearest-Neighborhood Clustering 准备。

[4]随机森林分类器的训练样本 training data 和测试样本 nXT (nX 代表负样本，T 代表 Testing)

A:测试样本 nXT

int half = (int)nX.size()*0.5f;

nXT.assign(nX.begin()+half, nX.end());即把 nX 中间到最后的数据赋值到 nXT 中，nXT 类型也是 vector<pair<vector<int>,int>>型，和 pX，nX 类型一致；

B:training data 的负样本 nX

nX.resize(half);即 nX 大小调整为原来 nX 的一半（19669 个），只保留前 half 个数据；

C:混合 nX 和 pX 成为最终的（既包含负样本又包含正样本）的 training data

（1）Vector<pair<vector<int>,int>> ferns_data(nX.size()+pX.size());

注意：ferns_data 大小是 19669+200=19899 个；

（2）Vector<int> idx（ferns_data.size()）；里面乱序保存 0 到 19899，涉及 random_shuffle 函数；

int a = 0;

（3）遍历 pX， ferns_data[idx[a]] = pX[i];a++;即把 pX 里面的 200 行元素，每行元素是 10*1 的 ferns 和对应的 1 给乱序的 ferns_data;

（4）遍历 nX，ferns_data[idx[a]] = nX[i]; a++;即把 nX 的 19699 个元素机器对应的 ferns 和 0 放到乱序的 ferns_data;

至此，随机森林分类器的 training data 搞定!

[5] NN 分类器的训练样本 training data 和测试样本 NNXT

A:测试样本 NNXT

half = (int)n_patch.size()*0.5f;

NNXT.assign(n_patch.begin()+half, n_patch.end());

B:training data

n_patch.resize(half); 即 n_patch 大小调整为原来 n_patch 的一半，只保留前 half 个数据;

Vector<Mat> nn_data(n_pattch.size()+1); //因为 p_patch 就一个

nn_data[0] = p_patch;

遍历 nn_data,顺序的将 n_patch[i]放入到 nn_data[1]到最后一个元素，因为第一个 nn_data[0]已经存储好了，而且不用乱序。至此，NN 分类器的 training data 搞定;

[6]分类器

Step 1: 准备 RF 分类器

```
int void Classifier::prepare(const vector<cv::Size> &scales) {
    int totalFeatures = nstructs*structSize;  //nstructs = 10; structSize = 13;
    features = vector<vector<Feature> >(scales.size(), vector<Feature>(totalFeatures));
    cv::RNG& rng = cv::theRNG(); //OpenCV自带的随机种子发生器
    float x1f, x2f, y1f, y2f;
    int x1, x2, y1, y2;
    //随机数去充填每一个尺度扫描窗口
    for (int i = 0; i < totalFeatures; i++) {
        x1f = (float)rng;
        y1f = (float)rng;
        x2f = (float)rng;
        y2f = (float)rng;

        for (int s = 0; s < scales.size(); s++) {
            x1 = x1f * scales[s].width;
            y1 = y1f * scales[s].height;
            x2 = x2f * scales[s].width;
            y2 = y2f * scales[s].height;
            features[s][i] = Feature(x1, y1, x2, y2);   //20行130列，元素是Feature(内含2对数值，共4个，代表像素位置)
        }
    }
    thrN = 0.5*nstructs; //随机森林分类时的阈值
    //完全二叉树，每层代表1个属性，structSize即13个属性，则树深13层，每层每个节点2片叶子，13层共8192片叶子
    //posteriors则为10棵树的后验概率，每棵树2^13次即8192片叶子;
    //pCounter为10棵树统计label为1的叶子，如果叶子全是正样本，则最多为8192，故空间开辟
```

```
8192个int值
    //nCounter为10棵树统计label为0的叶子，如果叶子全是负样本，则最多为8192，故空间开辟8192个int值
    for (int i = 0; i < nstructs; i++) {
        posteriors.push_back(vector<float>(pow(2.0, structSize), 0));
        pCounter.push_back(vector<int>(pow(2.0, structSize), 0));
        nCounter.push_back(vector<int>(pow(2.0, structSize), 0));
    }
}
```

Step 2: 提取 RF 特征 getFeature(Mat &patch, grid_idx, vector<int>& fern)

```
void Classifier::getFeatures(const cv::Mat &patch, const int &grid_idx, vector<int> &fern) {
    int leaf; //叶子，树的最终节点
    for (int t = 0; t < nstructs; t++) {//森林树的个数，共计10棵
        leaf = 0;
        for (int f = 0; f < structSize; f++) {  //对树上的属性操作，实现13层树的特征统计，leaf保存的特征
//struct feature特征结构体有一个运算符重载bool operator()
//返回的patch图像片在(y1,x1)和(y2,x2)点的像素比较值，返回布尔值
//leaf = 0 移位仍是0；不为0时，变为22,34,55,983...
//grid_idx传的是grid[idx].sidx,而它小于21个；
//每个scale有20行，每行13个Feature元素，grid[idx].sidx定了，就可确定目标属于哪个scale
            leaf = (leaf << 1) + features[grid_idx][t*nstructs + f](patch); //patch为grid[_idx]的图块
        }
        fern[t] = leaf;   //图块用基于类似LBP算法得出binary code(特征)保存在leaf里，该图块分10个特征，用fern保存，10行1列
    }
}
```

知识点：TLD的特征就是10个重叠度最好的grid对应的ROI，每个一次计算它们属于哪个尺度，然后计算13个属性后的leaf，将其存储到fern容器中。当10棵树都遍历结束，相当于把10个重叠度最好的grid对应的ROI全部提取特征，然后每10*1为一组，将其标记为1.

Step 3: RF 训练

A: 正样本总阈值和负样本总阈值

thrP = thr_fern*nstructs = 0.6*10 = 6；即每棵树为 0.6，总的为 10*0.6；

thrN = 0.5*nstructs = 0.5*10 = 5;

B：训练 training data

```
void Classifier::trainF(const vector<std::pair<vector<int>, int> > &ferns_data) {
    //所有树的正样本阈值之和
    thrP = thr_fern * nstructs;
    for (int i = 0; i < ferns_data.size(); i++) {
        if (ferns_data[i].second == 1) {//1表示正样本
            if (measure_forest(ferns_data[i].first) <= thrP)
//分类为正样本，但阈值却低于thrP时，该样本自加1，更新后验概率posterior为1
```

```
                update(ferns_data[i].first, 1, 1);
        }
        else {
                //分类为负样本，但阈值大于thrN,该样本需要判断pCounter[i][idx]是否等于0，如果为0，说明的确是负样本，则后验概率设置为0；如果该位置的pCounter不等于0，则负样本库+1，重新计算该位置的负样本概率（由于负样本库变大，最终导致分子小于分母，所以后验概率很大可能性小于thrN）
                if (measure_forest(ferns_data[i].first) >= thrN)  //thrN在tld.init()中的Classifier.prepare()算出,值为5（10棵树的总阈值10*0.5）
                update(ferns_data[i].first, 0, 1);
        }
    }
}
```

知识点：虽然重合度最高的 10 个 grid 被标记为 1 正样本，但每个 grid 算出的 RF 阈值（10 棵树的总阈值）却有可能低于 RF 阈值。为了纠正阈值错误，不影响模型的评估，故需要更新阈值。但更新前，要判断正样本计数器对应的标签。对正样本而言，如果阈值低于 thrP,该位置对应的 正样本计数器**一定**为 1，负样本计数器**一定**为 0，所以正确划分的正样本后验概率等于 pCounter/(pCounter+nCounter)=1/（1+0）=1；对负样本而言，如果阈值高于 thrN,则该位置对应的正样本计数器可能为 0，也可能不为 0.所以如果正样本计数器为 0，则该位置的后验概率一定为 0；如果正样本计数器不为 0，则 pCounter/（pCounter+(nCounter+1)）=0 即为后验概率。

Step 4: NN 训练

A：遍历 NN 的 traning data，对每个 training data 即包含 grid 的 ROI 放入在线模型，求相关和保守相似度（分别记为 rsConf 和 csConf）；

B：判断！如果样本为正样本（唯一的），其相关相似度 rsConf 小于 thr_nn（0.65），则推入 NN_pX 即真正的正样本；如果样本为负样本，其 rsConf 大于 0.5，则推入 NN_nX 即真正的负样本。

C：在线模型 onlineMode (Mat training data, vector<int>isin, float rsConf, float csConf)

（1） 创建 isin = vector<int>(3,-1),用来保存样本落入哪个库（主要分正样本库和负样本库）

（2） 判断：如果 NN_pX 是空的，则 rsConf 和 csConf 都为 0，函数返回；如果 NN_nX 是空的，则 rsConf 和 csConf 都为 1，函数返回。换句话说：正样本为空时，就只有负样本，则相关和保守相似度都应该为 0；负样本为空时，就只有正样本，则相关和保守相似度都应该为 1.

（3） 遍历 NN_pX，采用模板匹配 NN 分类器输入的训练数据

matchTemplate(NN_pX[i], training data（即 grid 的 ROI）, ncc, CV_TM_CCORR_NORMED)

对匹配结果 ncc 进行归一化矫正，进而得到归一化后的正样本概率 nccP，公式如下

$$\text{Similarity}(\text{Patch}_1,\text{Patch}_2) = (\text{NCC}_{\text{Patch}_1,\text{Patch}_2} + 1) * 0.5$$

$$\text{nccP} = ((\text{float}*)\text{ncc.data}[0] + 1) * 0.5$$

（4） 如果 nccP(图片对比的相似度)大于在线模型相似度的阈值(0.95),则 isin[0] = 1; isin[1]记录该阈值对应的 NN_pX 的地址；

（5） 遍历 NN_nX，重复（3-4），如果 nccN 大于在线模型相似度的阈值（0.95），则 isin[2]=1;不记录位置；

（6） 计算相关相似度和保守相似度，分别用csConf和csConf存储

```
//Measure Relative Similarity
float dN = 1 - maxN(即nccN);
```

```
float dP = 1 - maxP(即nccP);
rsConf = (float)dN / (dN + dP);
//Measure Conservative Similarity
dP = 1 - csmaxP(即nccP);
csConf = (float)dN / (dN + dP);
```

总之，在线模型用的是已有的正负样本和 training data 匹配，如果样本重合度大于 0.95，则判断出他们是否属于正负样本。

[7]分类器评估初始模型

```
evaluateTh(const vector<pair<vector<int>, int> > &nXT, const vector<cv::Mat> &nExT)
```

（1） 评估随机森林分类器

```
fconf = (float)measure_forest(nXT[i].first) / nstructs;
if (fconf > thr_fern)  //thr_fern=0.6
    thr_fern = fconf; //更新单棵树的阈值
```

（2）评估NN分类器

```
vector <int> isin;
float conf, dummy;
for (int i = 0; i < nExT.size(); i++) {
    onlineNNModel(nExT[i], isin, conf, dummy);
if (conf > thr_nn)
    thr_nn = conf;
}
if (thr_nn > thr_nn_valid) //检测器检测与跟踪器预测的box的重叠度阈值，0.7
    thr_nn_valid = thr_nn;
```

总之，分类评估初始模型是为了确定 RF 和 NN 分类器的阈值。将来有新目标作为测试数据时，必须高于这两个分类器的阈值，才能通过 RF 和 NN 分类器。

TLD Process 模块

[1] 跟踪模块

A:跟踪

应用中值流 median-flow tracker，主要步骤是【1】在当前灰度图均匀撒点，产生 10*10 个待跟踪点；【2】光流法计算 pregray->nowgray 的 pts1 到 pts2，和 nowgray->pregray 的 pts2 到 pointsFB;【3】遍历 pts1,计算 FB_error[i] = norm(pointsFB[i] - pts1[i]);【4】归一化处理解相似度，根据正向光流法的 status,裁剪以 pointsFB [i]和 pts1[i]为中心的 rect0, rect1,计算相似度（和在线模型用的模板匹配计算 nccP 方法一致）;【5】筛选：根据相似度求 median,凡是相似度大于 median 的，保留 pts1 和 pts2 的点，再根据反向光流得到 FB_error 的 median, 凡是 FB_error 小于 median 的 pts1[i]和 pts2[i]都保留。最后重新整理 pts1 和 pts2 即可，注意二者 size 相同;

B:根据跟踪点画出目标位置

假设 pts1 和 pts2 只剩 8 个点，显然点的轮廓不一定有规则，怎么选定目标区域呢？主要步骤是【1】计算相邻两帧像素的偏移量：pts2[i]-pst1[i]的 xoff[i]和 yoff[i]，利用中值 dx=median(xoff)和 dy=median(yoff)，得到点在相邻两幅灰度图中的的偏移量 dx 和 dy；【2】遍历这 8 个点，并嵌套 j=i+1,依次计算 d.push_back(norm(pts2[i] - pts2[j]) / norm(pts1[i] - pts1[j]))；大小是 d.reserve(npoints*(npoints - 1) / 2)=28；计算 s= median(d);根据下面代码，计算出临时的目标框。如果跟踪失败，只有 1 个光流或者 0 个光流跟踪到，则 s=1;可见下面的思路就是利用中值法去掉边长过大或者距离过于突兀的点。

```
float s1 = 0.5*(s - 1)*lastbox.width;
```

```
float s2 = 0.5*(s - 1)*lastbox.height;
tbb.x = round(lastbox.x + dx - s1);
tbb.y = round(lastbox.y + dy - s2);
tbb.width = round(lastbox.width*s);
tbb.height= round(lastbox.height*s);
```

C:getPattern 即得到模式

【1】 假设跟踪成功，先判断 tbb 是否越界；不成功则标记 tvalid = false,后续不启动学习；

【2】 取样式：getPattern(nowgray(tbb),tpattern,tmean,tstdev)； //缩放到 15*15 的 patch

D:跟踪结果与在线模型对比求相关和保守相似度

与在线模型对比，求解相似度 trsConf 和 tcsConf，如果保守相似度 tcsConf 大于 NN 分类器模型阈值，则跟踪判定为有效，tvalid=true,为后续启动学习模块做参考。如果小于或等于 NN 分类模型的阈值，则 tvalid=false,不启动学习模块.

总之：跟踪模块主要是在原始目标框内生成 100 个特征点，然后根据跟踪结果找到当前帧中的位置，然后框住。同时，跟踪模块给出它的相关和保守相似度,以及是否启动学习的标识 tvalid.

[2]检测模块

A：对 nowgray(tbb)进行积分图求解（为了求方差），接着高斯模糊（提高相似度对比）；

B：方差和 RF 分类器筛选：遍历 grid,以每个 grid 对应的图块在积分图的位置上求方差，如果该值大于 best_box 对应图块的方差值，则用 RF 提取特征并计算 10 棵树的投票。如果投票阈值 conf（模型训练时的后验概率）大于 RF 分类器模型的阈值，则保存该网格的索引号到 dt 中；如果方差小于原始目标的方差，则投票阈值为 0.

C：NN 分类器的筛选：通过方差分类和 RF 后，对通过的 grid 根据投票阈值排序，取前 100 个图块提取样式，即 getPattern（patch, dt.patch[i],mean,stdev）;接着把 dt.patch[i]放到在线模型，计算出 dt.rsConf, dt.csConf。如果 dt.rsConf 大于 NN 分类器模型的阈值,则将该 patch 对应的 grid[i]和保守相似度 dt.csConf 放到通过 TLD 检测模块的结果中 dbb 和 dconf。如果 dbb 的大小不为 0，则检测标识为成功。

总之，检测模块是根据初始化的全局网格在以当前跟踪结果为中心的 ROI 上进行方差，RF 和 NN 分类器的筛选，凡是通过三个分类器的，记录该网格的相关和保守相似度，赋予检测模块成功标识。注意，之所以用当前 tbb 为中心的 ROI，是因为考虑相邻两帧差别不大，目标没有超出图片的视野。

[3]综合模块

【1】判断是否跟踪成功，如果没有，则 tvalid=false，不开启学习，且使用通过检测模块的 grid 为下一帧初始目标位置。考虑检测模块可能有多个潜在目标框，需要对这些框进行聚类分析，主要是利用 partion 函数（树深和根）将多个框依据重叠度合并为 1 个或 2 个，保存在 cbb（current bounding box）中，用的时候默认用 cbb[0]第一个元素；

【2】如果跟踪成功，再判断是否检测成功。如果检测不成功，直接返回 tbb 作为下一帧新位置；

【3】如果检测成功，则通过重叠度对检测器检测到的 grid 即 dbb 和 dconf 进行聚类分析，得到 cbb 和 cConf。计算 cbb 中每个框与 tbb 的重合度。如果重合度小于 0.5 但聚类得到的保守相似度大于跟踪得到的保守相似度，则置信度计数器加 1 且记录该 cbb[i]的索引 idx；否则什么也不做；

【4】如果置信度计数器等于 1，则下一帧的框位置即为 cbb[idx],tvalid = false 不开启学习；说明凡是检测器检测得到的 dbb 是不用重新学习的，因为跟踪才有背景的变化；

【5】如果置信度计数器不等于 1,则遍历 dbb,求 tbb 和 dbb 重合度大于 0.7 的位置,保存为 cx,cy,cw,ch,并且 close_detection+1.然后分大于 0 和小于等于 0 的情况。如果大于 0，下一帧位置则等于

```
bbnext.x = cvRound((float)(10 * tbb.x + cx) / (float)(10 + close_detections));
bbnext.y = cvRound((float)(10 * tbb.y + cy) / (float)(10 + close_detections));
bbnext.width = cvRound((float)(10 * tbb.width + cw) / (float)(10 + close_detections));
```

bbnext.height = cvRound((float)(10 * tbb.height + ch) / (float)(10 + close_detections));
如果小于等于 0，则说明检测器没有在 tbb 附近检测出目标，什么也不做，返回 tbb 作为下一帧位置。总之，综合模块是为了确定如何选择检测模块和跟踪模块的潜在目标框作为下一帧位置。第一种情况是如果检测框的保守相似度高，但和跟踪框重合度低，则直接用检测框。如果检测框是多个，则通过权值矫正，让框的位置居于中心，如果框唯一，则直接用该检测框；第二种情况，如果检测框的保守相似度低，但和跟踪框重合度高，则建立检测模块框和跟踪模块框的矫正关系，进行矫正。原理就是(tbb*矫正因子+跟踪框计算的矫正值)/(矫正因子+close_detection);第三种情况，是如果检测框保守相似度低于跟踪框保守相似度，二者重合度还不高，则默认选择跟踪框作为下一帧位置。

知识点：

相关相似度

$$rsConf = \frac{P^+}{P^+ + P^-}$$

保守相似度

$$csConf = \frac{P^+_{50\%}}{P^+_{50\%} + P^-}$$

P^+代表正样本，P^-代表负样本，$P^+_{50\%}$代表正样本库中相似度排名前 50%的正样本.注意跟踪框严格参照上述公式计算相关和保守相似度，检测模块同样利用上面的公式，不过是靠取 NN_pX 的一半数据来。实际上 NN_pX 只有 1 个，所以作者用了 ceil(NN_pX.size()*0.5)来做。

[4]学习模块

前提：tvalid 标识必须为真，才可以启动学习模块。且学习模块在跟踪，检测，综合模块之后。

【1】提取样式+在线模型对比

getPattern(nowgray(bbnext), learn_pattern, learn_mean, learn_stdev); //缩放到 15*15

classifier.onlineNNModel(learn_pattern, learn_isin, learn_conf, learn_dummy);

【2】判断是否学习

(1)相关相似度阈值判断：learn_conf < 0.5; //变化太快，不学习;

(2)方差判断：pow(learn_stdev.val[0], 2) < var; //方差过小，不学习；注意在 generateNegativeData 时，为了过滤不必要的负样本，用的是 0.5*var；而检测模块用的方差分类器是 getVar(…)>=var;学习却用的小于 var 来判断，注意区别！

(3) 正负样本判断：learn_isin[2]==1; //当前目标根据在线模型对比，发现是负样本，则不学习;

【3】需要学习

第一步：清空 good_boxes 和 bad_boxes;

第二步：遍历 grid，求解 grid[i].overlap = bbOverlap(bbnext, grid[i]); //为 getOverlapBoxes 准备;

第三步：计算新的 good_boxes, bad_boxes, best_box(getOverlapBoxes(grid, 10));

第四步：生成正样本库 generatePositiveData: 如果 good_boxes.size()>0. 生成正样本 pX；否则不学习，因为没有 good_boxes;

第五步：特征提取+生成 RF 训练所需的 training data(fern_examples)

主要原理是混合 pX+bad_boxes;遍历 bad_boxes,把检测模块过 RF 分类器提取特征 getFeatures 得到的 fern 和 measure forest 得到的投票阈值>=1 的样本推入到新的训练数据 fern_examples，如下所示;

```
for (int i = 0; i < bad_boxes.size(); i++) {
        idx = bad_boxes[i];
        if (tmp.conf[idx] >= 1) {
            fern_examples.push_back(make_pair(tmp.patt[idx], 0));
```

```
        }
}
```

第六步：生成 NN 训练所需的 training data(nn_examples)

```
vector<cv::Mat> nn_examples;
nn_examples.reserve(dt.bb.size() + 1);  //通过检测模块的框放入训练数据，长度额外加1
nn_examples.push_back(p_patch); //加 1 的目的是能够推入 p_patch
```

遍历检测框（通过三个分类器后也就 1,2,3 或 4 个），依次计算检测框和 bbnext 的重合度，如果重合度小于 bad_overlap(0.2)，则将该检测框推入 nn_examples;

第七步：训练 RF 和 NN 分类器

trainF(fern_examples)和 trainNN(nn_examples)；目的是更新后验概率，正样本计数器和负样本计数器。

总结：学习模块（1）清空 good_boxes 和 bad_boxes;（2）遍历网格重新生成 good_boxes 和 bad_boxes 以及 best_boxes;（3）根据新网格，生成新的混合正负样本的 RF 训练数据；（4）RF 特征提取，计算投票阈值，训练 RF 分类器；（5）利用原始最好的 p_patch 和检测框组成 NN 训练数据，训练 NN 分类器。在线模型的 RF 阈值和 NN 阈值还是用的初始化的结果，学习后并不改变这些。学习模块只改变后验概率，以及模型的正负样本计数器。特别是对于 NN 分类器而言，原来样本计数器是 50 个，现在就 5 或 6 个（可以改进的点）。网上流传的样本库不断膨胀是错误的！每当要学习的时候，总会根据新的 good_boxes, bad_boxes 产生新的重合度，特征提取，RF 投票值，进而 RF 训练后产生新的后验概率，pCounter 和 nCounter.

TLD 有关疑问

1 为什么选随机蕨+NN 分类器

答：

2 训练数据为什么不选其他训练方法产生

答：训练方法根据训练数据有无标签分为监督学习和半监督学习。

监督学习分类方法有：逻辑回归，决策树，KNN，朴素贝叶斯等针对二分类方法。

半监督分类方法有：BP 神经网络，SVM，深度学习等抗噪声干扰能力强的算法。

训练方式有：有多示例训练(半监督学习方法的衍生版本)，协同训练，自助采样（Boostrapping），提升法(Boosting)，自训练模型（self-training），期望最大化（Expectation-Maximum）

[1]自助采样

(1)在数据库中随机抽取一部分数据;

(2)利用该数据训练分类器;

(3)分类器训练好后，利用其测试数据库中除该部分训练数据意外的数据（测试数据）;

(4)将测试数据分类错误的数据加入原始那部分训练的数据;

(5)不断重复(2-4), 直到全部数据分类正确。

TLD 采用的类似自助采样法，区别在于有 P-N 学习法进行错误样本的矫正。

[2]提升法

(1)数据库中抽取 n 个数据，权值为 1/n，并分别用几个弱分类器训练;

(2)在上述权值下，挑选当中最好的分类器（loss function 值最小）并更新分类错误的样本的权值（分类错误的样本的权值一般增大，即今后训练给予更多关照）;

(3)在第二步基础上继续用几个分类器训练，找到最好的分类器，然后给予该分类器对应的预测函数权值;

(4)重复第二步到第三步，直到找到带有不同权值的 H1, H2, …，Hn(预测函数);

(5)最终分类器是弱分类器的组合，例如 f = H1*a1 + H2*a2 + H3*a3…

提升法的典型代表就是 Standard AdaBoost 和 Cascaded AdaBoost。TLD 不用该方法是因为 TLD

滑动窗口过多，权值不好分配，训练时间过长，不利于在线训练的效率；

[3]多视角协同训练（主要是二分类问题）

(1)在已标记的数据集 L 上取 x1 和 x2 训练数据（视角指根据不同属性如颜色和梯度属性划分），准备两个分类器 f1 和 f2；

(2)从未标记数据集 U 上随机的选取 m 个示例放入集合 U'中;

(3)用已标记的数据集 L 的 X1 部分训练出一个分类器 f1;

(4)用已标记的数据集 L 的 X2 部分训练出一个分类器 f2;

(5)用 f1 对 U'中所有元素进行标记，从中选出 p 个正标记和 n 个负标记;

(6)用 f2 对 U'中所有元素进行标记，从中选出 p 个正标记和 n 个负标记;

(7)将上面选出的 2p+2n 个标记加入 L 中;

(8)随机从未标记的 U 中再选取 2p+2n 个数据补充到 U'中，重复(3-7)直到 U 被清空；

注意：f1 和 f2 可以是同一种分类器也可以是不同分类器；p 和 n 可以相同，也可以不同。

[4] Self-Training Models 自训练模型

假设我有两堆数据 A 和 B，其中 A 是已标注的数据，即带 Label 的；而 B 是未标注数据。Self-Training 的做法如下：

（1）从已标注数据 A 中训练一个分类模型 M

（2）用该模型对 B 进行预测

（3）将预测结果中置信度高的 K 个样本，连同它们的 Label 加入训练数据 A，并从 B 中删除

（4）回到第 1 步。

注意：第 3 步可以有很多实现方法，比如可以将 B 中所有样本加入 A，根据预测时置信度的不同给样本赋予不同的权重。

[5] 期望最大化

A、B 两者都是不知道的，但是假设我们在知道了 A 的信息之后，就可以直接求得 B 的信息，反过来也是一样的，知道 B 也可以直接求得 A。即首先对两者中的任何一个赋予某种初值，从而可以得到另一个的估计值，然后由得到的估计值再去计算另外一个的值，然后对两个值的错误率设置一个阈值或者停止条件，直到收敛停止这个过程，得到我们所要求得的两个值 A 和 B。其中涉及初始化模型参数，收敛判定（Jensen 不等式）。

[6]多示例训练

原理是：数据有标记，但标记的目标不是一个样本，而是一个数据包（bag）。当一个 bag 的标记为负时，这个 bag 里面所有样本的标记都是负的。当一个 bag 的标记为正时，这个 bag 里面至少有一个样本的标记为正。 我们的目标是学习得到一个分类器，使得对新输入的样本，可以给出它的正负标记。这样的一类问题就是多示例问题。

过程是不清楚，没搜到相关答案。

3 检测模块检测原理来自于哪儿

答：检测目标原理主要分图片

[1]图片特征检测

方法是：哈里斯角点，图片像素点，Fast 特征点检测；

[2]目标实例检测

方法是:(1)全局模型，即把同一目标进行不同角度拍照，保留这些实例。跟踪时，凡是和实例库相似度高的目标可判定为要找的目标；(2)本地模型，利用的是 DOG 高斯不同原理；(3)学习模型，即利用机器学习方法，特征提取，模型评估。三个方法都有缺点：第一个是背景有遮挡的图片无法有效监测；第二个是不适合运动场景，相邻帧很近，DOG 可能差别不大，此外计算效率低；第三个很好，较成熟，但 TLD 之前用的少。

[3]图像编码特征

方法是：类似 haar 特征人脸检测，具体看图像处理面试题目关于 haar-like 人脸检测原理。

4 检测模块在 TLD 之前有什么问题

答：检测模块主要是[1]offline 检测，且检测和跟踪运行时是分开的；[2]TLD 之前用的都是监督学习，半监督学习主要用在文字分类。TLD 之后才有了半监督学习在目标跟踪领域的应用，主要原因是半监督学习分类时很强调目标的独立特征属性。

5 跟踪模块在 TLD 之前有什么问题

答：[1]光流法的缺点：

(1) 容易因位置错误的积累导致漂移，最终导致跟踪失败;(2)如果突然发生形变，遮挡，消失则易丢。

[2] Median-Flow Tracker 缺点

同光流法类似，虽然强于光流法，降低了位置累积的错误，但一旦有遮挡，很容易跟丢。

6 为什么 TLD 不用 AdaBoost

答：AdaBoost 的设计初衷是针对训练数据少的样本。原理在于其预测函数和权值

$$h(x)=\frac{1}{2}\log(\frac{P_D(y=1|x)}{1-P_D(y=1|x)})$$

$$H=\text{argmin}Z_D(h)=\sum_{i=1}^{m}(d_i e^{-y_i h(x_i)})$$

P_D是根据当前数据的权值计算的概率；$Z_D(h)$是指数损失函数；d_i是数据的权值，需要$d_i^{t+1}=\frac{d_i^t e^{\alpha}}{\text{sum(D)}}$更新。可见如果样本量太大，上述两个公式涉及的权值系数怎么赋值和弱分类器训练都不高效。

7 跟踪失败如何判定

答：[1]像素做 Sum of Squared Difference(SSD)，即基于两点的轨迹，计算 Forward-Backward Error SSD。值越大，结果越差；TLD 用到了 NCC，FB_error 和 SSD 三个。

8 什么是 Quasi-random Weighted Sampling (QWS)?以及采样时的数据无偏？

答：QWS 中文翻译为准随机加权采样，即从 m 个数据库中选出 n 个样本。后者的样本期望，方差和前者一致，则被称为数据无偏，否则数据有偏倚，模型不准。TLD 用到的将前 10 个正样本复制 20 次就是这个这个理念，但这样做的缺点是 n 很大时，浪费内存。

9 为什么负样本的产生 nX 的 patch 方差大于 0.5*var；而检测模块通过方差分类器是大于等于 var？

答：[1]全局网格中，nX 本身计算时按照 patch 大于 0.5*var，说明凡是低于 0.5*var 的 nX 有可能包含目标，而高于此值的一定不包含目标。只有这样，去掉的 nX（包含潜在正样本的）不会和 pX（真正正样本）重复计算，从而减少计算量；[2]正样本方差一定大于或等于 var,所以最终检测模块会有检测到的目标。

10 TLD 无法应付突然变向的目标，怎么办？

答：一次运行多个 tracker，然后用多个 P-N 专家学习和训练，增强模型的泛型能力。

11 P-N 学习说一下

答：P-N 学习本质很简单，是基于观察的基础上发展而来。只在初始化训练数据和学习模块里进行诊断，跟踪模块+检测模块+综合模块并没有使用 P-N 学习。原理公式：

正常情况下：

$n^+(k+1)=n_f^+(k)+n_c^+(k)$ （1）

$n^-(k+1)=n_f^-(k)+n_c^-(k)$ （2）

$n^+(k+1)$代表迭代次数在$k+1$时的正样本，n_f^+代表错误分类的正样本，n_c^+代表正确分类的正样本；

$FP(k+1)=FP(k)+n_f^+(k)-n_c^+(k)$ （3）

$FN(k+1)=FN(k)+n_f^+(k)-n_c^+(k)$ （4）

$FP(k+1)$代表迭代次数在$k+1$时的错分为正样本的误差。

12 准确率，精确率，召回率，F1 score

答：这三个指标是衡量分类器性能的指标，

TP：实际为正样本，预测为正样本；真正的正样本

FP：实际为负样本，预测为正样本；错误的正样本

TN：实际为负样本，预测为正样本；真正的负样本

FN：实际为正样本，预测为负样本；错误的负样本

假设总样本数 100，现取出一部分 50 个子样本作为测试样本，其中真正的正样本数是 40，真正的负样本是 10；错误的正样本是 30，错误的负样本是 20；则

准确率：(TP+TN)/总样本数= (40+10)/100 = 50%;

精确率：TP/（TP+FP）=40/(40+20)=66.67%

召回率：TP/(TP+FN)=40/70=57.14%

F1 = 2*(精确率*召回率)/(精确率+召回率)=2*0.6667*0.5714/（0.6667+0.5714）=61.53%

简而言之，东西是不是某个类的时候，准确率就是它说是，这东西就确实是的概率吧，召回率就是，它说是，但它漏说了（1-召回率）这么多。注意：准确率和召回率是互相影响的，理想情况下肯定是做到两者都高，但是一般情况下准确率高、召回率就低，召回率低、准确率高，当然如果两者都低，那是什么地方出问题了。F1 是精度和召回率的结合，方便迅速评估。

13 重合度，均匀撒点，根据点画框，外接矩形，中值计算，位置矫正 cbb

【1】重合度(假设两个 Rect 型 box1，box2)

第一步：判断是否相交

```
box1.x > box2.x + box2.width;
box1.y > box2.y + box2.height;
box2.x > box1.x + box1.width;
box2.y > box1.y + box1.height;
```

第二步：计算重合矩形的宽和高

```
float W = min(box1.x + box1.width, box2.x + box2.width) - max(box1.x, box2.x);
float H = min(box1.y + box1.height, box2.y + box2.height) - max(box1.y, box2.y);
```

第三步：计算重合度

```
float intersection = W * H;
float area1 = box1.width * box1.height;
float area2 = box2.width * box2.height;
float overlap = intersection/(area1+area2-intersection);
```

【2】根据 lastBox，如何撒点，以便光流法跟踪

第一步：网格边界和步长值

```
int margin_h = 0; //横向采样边界
int margin_v = 0; //纵向采样边界
int max_pts  = 10;//patch 里 10*10 撒点方式
int stepx = ceil(double((box.width - 2 * margin_h) / max_pts));
int stepy = ceil(double((box.height - 2 * margin_v) / max_pts));
```

第二步：撒点（必须要有个 box）

```
for (int y = box.y + margin_v; y < box.y + box.height - margin_v; y += stepy) {
        for (int x = box.x + margin_h; x < box.x + box.width - margin_h; x += stepx) {
            pts1.push_back(cv::Point2f(x, y));
        }
    }
```

【3】根据点画框（必须有 vector<Point2f> pts1 和 pts2）

第一步：计算 pts1 到 pts2 的点在 x 和 y 方向上的偏移

```
for (int i = 0; i < pts1.size(); i++) {
        xoff[i] = pts2[i].x - pts1[i].x; //x方向
        yoff[i] = pts2[i].y - pts1[i].y; //y方向
    }
```

第二步：计算偏移的中值

```
float dx = median(xoff); //x方向偏移的中值
float dy = median(yoff); //y方向偏移的中值
```

第三步：计算pts2内部点之间距离与pts1内部点之间距离的比值,该比值就是放缩尺度

```
if (npoints > 1) {
    vector<float> d;
    d.reserve(pts1.size()*(pts1.size() - 1) / 2); //100个点经过中值流只剩8个
    for (int i = 0; i < pts1.size(); i++) {
        for (int j = i + 1; j < pts1.size(); j++) {
            d.push_back(norm(pts2[i] - pts2[j]) / norm(pts1[i] - pts1[j])); //总长度28
        }
    }
    s = median(d); //该比值求中值
  }else
        s = 1.0; //如果只有一个跟踪点，则新位置等同于旧位置
```

第四步：根据lastBox和放缩尺度确定新的box

```
float s1 = 0.5*(s - 1)*lastbox.width;
float s2 = 0.5*(s - 1)*lastbox.height;
tbb.x = round(lastbox.x + dx - s1);
tbb.y = round(lastbox.y + dy - s2);
tbb.width = round(lastbox.width*s);
tbb.height= round(lastbox.height*s);
```

【4】多个框，如何外接矩形

```
bbhull.x = min(box1.x, box2.x);
bbhull.y = min(box1.y, box2.y);
bbhull.width = max(box1.x+box1.width, box2.x+box2.width);
bbhull.height= max(box1.y+box1.height, box2.y+box2.height);
```

【5】中值计算

给定一 vector<float> v;

```
int n = floor(double(v.size() / 2));
//比v[n]大的排后面，小的排前面，左右不一定有序
nth_element(v.begin(), v.begin() + n, v.end());
return v[n]; //注意返回值，是一个 float 型的变量，元素在 v 中的第 n 个位置
```

【6】框聚类合并

第一种情况：框就一个

答：就不用聚类了，直接返回该框；

第二种情况：框有两个

重合度大于 0.5：

```
box.x = (box1.x + box2.x)/2;
```

```
box.y = (box1.y + box2.y)/2;
box.width = (box1.width + box2.width)/2;
box.height= (box1.height+box2.height)/2;
```

重合度小于 0.5：

情况和第三种情况相同，可以和第三种情况用同一种方法，也可以分开想 TLD 作者那样，可以加速运算；

第三种情况，框有多个，可能好几类

```
vector<int> T;
T = vector<int>(dbb.size(), 0);
flag = partition(dbb, T, *bbcomp); //T 的 1,2,3...代表树的根节点是否相同，返回的是聚类个数
//bool bbcomp(box1, box2){ if bbOverlap(b1,b2)<0.5 return false; else return true}
for (int i = 0; i < flag; i++) { //flag值几个框
        int N = 0, mx = 0, my = 0, mw = 0, mh = 0;
        for (int j = 0; j < T.size(); j++) {
            if (T[j] == i) {
                mx = mx + dbb[j].x;
                my = my + dbb[j].y;
                mw = mw + dbb[j].width;
                mh = mh + dbb[j].height;
                N++;
            }
        }
        if (N > 0) {
            bx.x = cvRound(mx / N);
            bx.y = cvRound(my / N);
            bx.width = cvRound(mw / N);
            bx.height = cvRound(mh / N);
        }
    }
```

解析：假设有三个聚类，每个聚类分别有 3 个，5 个，10 个框。由于 partition 会赋予 T 中的标签，会导致每次遍历 T 中的标签时，只有标签一致的才回累加求平均值，进而计算出对应标签该聚类的位置。Partition 是 opencv 内置函数。

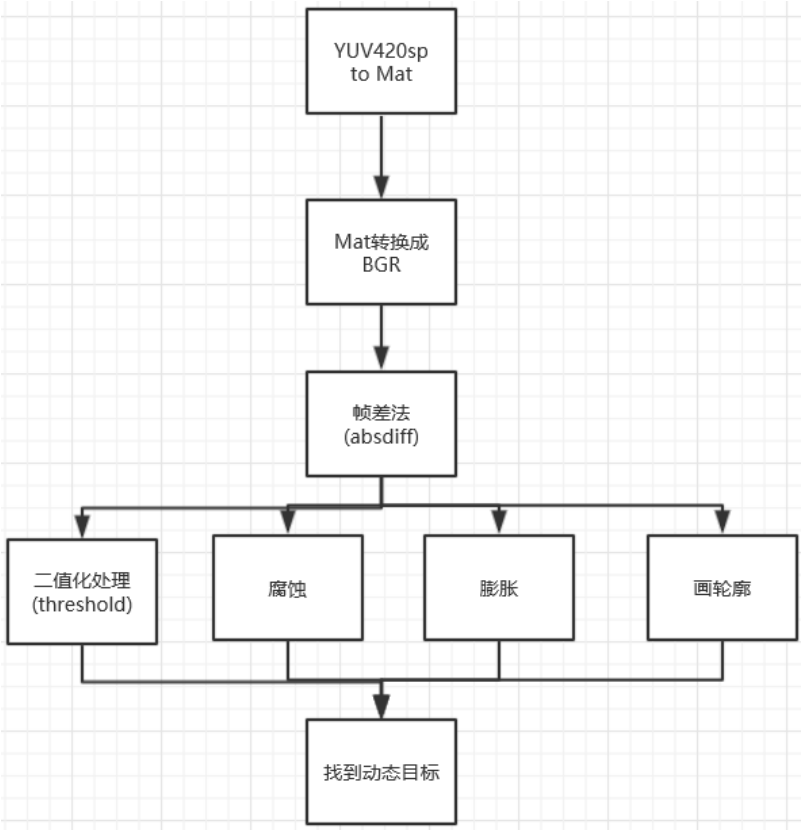
14 相机坐标系和像平面坐标系如何转换

答：图像坐标系分像素坐标系和物理坐标系。前者单位是像素，后者单位是毫米。比像素坐标系更接近实际的是像平面坐标系。像平面坐标系在云台上是以图像中心为基准，划分四个像平面，上下左右正负 200 或 2000，由云台决定。代码实现如下：

```
int sx = (bbox.x+bbox.width/2)*2000/160 - 1000;
int sy = (bbox.y+bbox.height/2)*2000/90 - 1000;
```

运动目标检测

1 流程



2 API

[1]Mat **getStructuringElement(**int shape, Size ksize, Point anchor = Point(-1,-1));

Shape：指结构元的形状，MORPH_RECT 矩形，MORPH_CROSS 十字型，MORPH_ELLIPSE 椭圆；

Ksize：指结构元的大小，一般为奇数，如 3x3, 5x5, 7x7，从而有中心和半径的概念；

Anchor：锚点位置，如 3x3 的锚点为位置在中心，设置为-1,-1 即默认位置，否则有偏移；

例如：Mat kernel = **getStructuringElement(**MORPH_RECT, Size(3,3));

[2] **erode**(source, image, kernel);

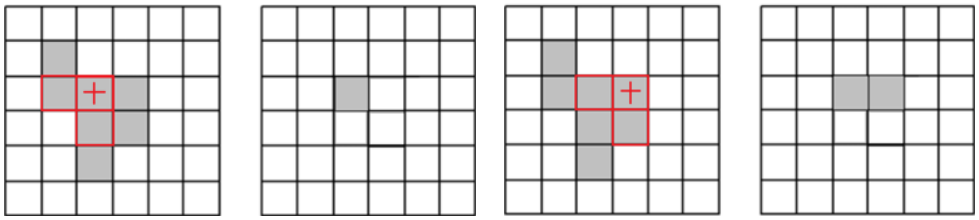
Source: 经过 threshold 处理的二值化图；

Image：腐蚀操作后的结果图，kernel 是[1]中的例子；

腐蚀原理：

A: 在原图中找出第一个和结构元完全匹配的部分；

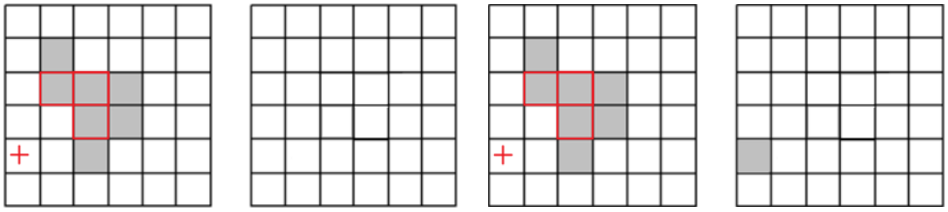
B: 拿结构元扫描二值化图像，和结构元每个元素都匹配时，则保留结构元掩模下锚点位置对应的像素值；否则删除该锚点对应位置处的原像素；



第一次处理　　第二次处理

C：重复 A-B，直至结束。

注意分两种情况，结构元锚点默认在中心（如 B 所示）和锚点带偏移的情况。如下图示：



处理前　　处理后

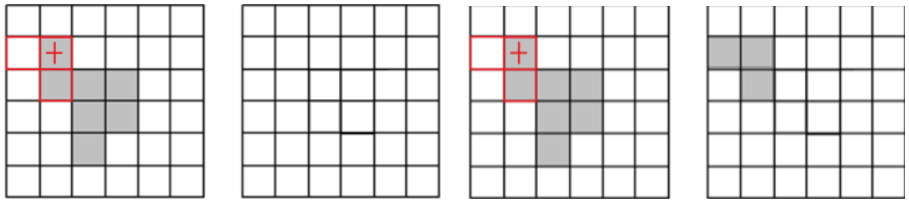
[3] **dilate**(source, image, kernelDilate);

参数同 erode 一样，不同点在于 kernelDilate 可能是 5x5,7x7 等，用户确定

膨胀原理：

A：在原图中找出第一个像素值，然后和结构元锚点进行匹配；

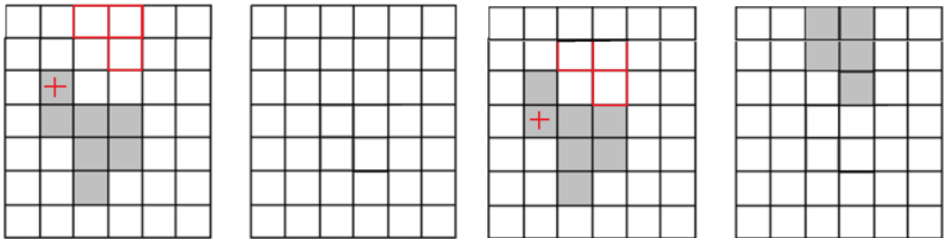
B：如果锚点和该位置像素值相同，则结构元掩模下每个元素用锚点对应的原像素值替代；否则不做任何操作；



处理前　　处理后

C：接续第二个像素值，重复 B，直至结构元扫描完图像。

类似腐蚀，也分锚点在结构元上还是结构元外，如图示：



第一次处理　　第二次处理

[4] findContours 详解

原型：findContours(InputOutputArray image, OutputArrayOfArrays contours, OutputArray hierarchy, int mode, int method, Point offset=Point());

Image:单通道图像，可以是灰度图，一般最好是二值图，效果更好；

Contours：轮廓，vector <vector<point>>型，contours.size()计算轮廓数量，而 contours[i].size()计算第 i 个轮廓上点的数量；

Hierarchy： vector<Vec4i> 型，即向量模板类定义的只有 4 个整型元素的向量；hierarchy[i][0] ~hierarchy[i][3]，分别表示第 i 个轮廓的后一个轮廓、前一个轮廓、父轮廓、内嵌轮廓的索引编号。如果当前轮廓没有对应的后一个轮廓、前一个轮廓、父轮廓或内嵌轮廓的话，则 hierarchy[i][0] ~hierarchy[i][3]的相应位被设置为默认值-1；

Mode：取值为整型，定义轮廓的检索模式，如只显示外部轮廓，内外都显示等；

Method：定义计算轮廓的近似方法；

Offset：指偏移量，所有的轮廓信息相对于原始图像对应点的偏移量，相当于在每一个检测出的轮廓点上加上该偏移量，并且 Point 还可以是负值！

[5] void **drawContours**(Input image, input contours, int contour id, const Scalar& color, int thickness=1, int lineType=8, vector<Vec4i> hierarchy, int maxLevel=INT_MAX, Point offset=Point())

第一个参数 image 表示目标图像，

第二个参数 contours 表示输入的轮廓组，每一组轮廓由点 vecto<vector<Point>>构成，

第三个参数 contour id 指明画第几个轮廓，如果该参数为负值，则画全部轮廓，

第四个参数 color 为轮廓的颜色，

第五个参数 thickness 为轮廓的线宽，如果为负值或 CV_FILLED 表示填充轮廓内部，

第六个参数 lineType 为线型，

第七个参数 hierarchy 为轮廓结构信息. 后续略。

[6] 轮廓包裹 boundingRect, minAreaRect, minEnclosingCircle

A: Rect boundingRect(points)

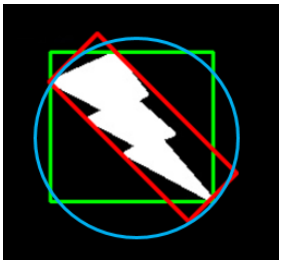
输入为包含点的容器(vector<Point>型)；返回的是最小矩形；

B: RotatedRect minAreaRect(points)

输入同上，返回的是 RotatedRect 型矩形；

C： void minEnclosingCircle(points, Point2f& center, float& radius)

输入第一个参数同上，第二个是圆心，第三个是半径；返回类型为空。



人脸检测

opencv 声明类时，为什么前面有一个 CV_EXPORTS 修饰符?

答：CV_EXPORTS 的作用是若要导出类中的所有公共数据成员和成员函数，关键字必须出现在类名的左边。由于对名称修饰没有标准规范，因此导出函数的名称在不同的编译器版本中可能有所变化。如果使用 #define CV_EXPORTS _declspec(dllexport)，即实际执行的 __declspec(dllexport)，是为了防止编译出错，仅当解决任何命名约定更改时才必须重新编译 DLL 和依赖 .exe 文件。

1 流程

见：人脸检测+跟踪流程图

2 API

//模板匹配

Mat image_matched;

matchTemplate(image_source, image_template, image_matched, TM_CCOEFF_NORMED);

模板匹配即所谓的 pixel by pixel 匹配，最后一个参数决定匹配方式，共

A: TM_SQDIFF：该方法使用平方差进行匹配，因此最佳的匹配结果在结果为 0 处，值越大匹配结果越差；

B: TM_SQDIFF_NORMED：该方法使用归一化的平方差进行匹配，最佳匹配也在结果为 0 处；

TM_CCORR：相关性匹配方法，该方法使用源图像与模板图像的卷积结果进行匹配，因此，最佳匹配位置在值最大处，值越小匹配结果越差；

C: TM_CCOEFF_NORMED：归一化的相关性匹配方法，与相关性匹配方法类似，最佳匹配位置也是在值最大处，值越小匹配结果越差。

//寻找最佳匹配位置

double minVal, maxVal;

Point minLoc, maxLoc;

minMaxLoc(image_matched, &minVal, &maxVal, &minLoc, &maxLoc);

知识点：由于使用 TM_CCOEFF_NORMED，故要找匹配值最大的，所以标准就是 maxVal 以及其对应的 maxLoc（目标所在位置）。

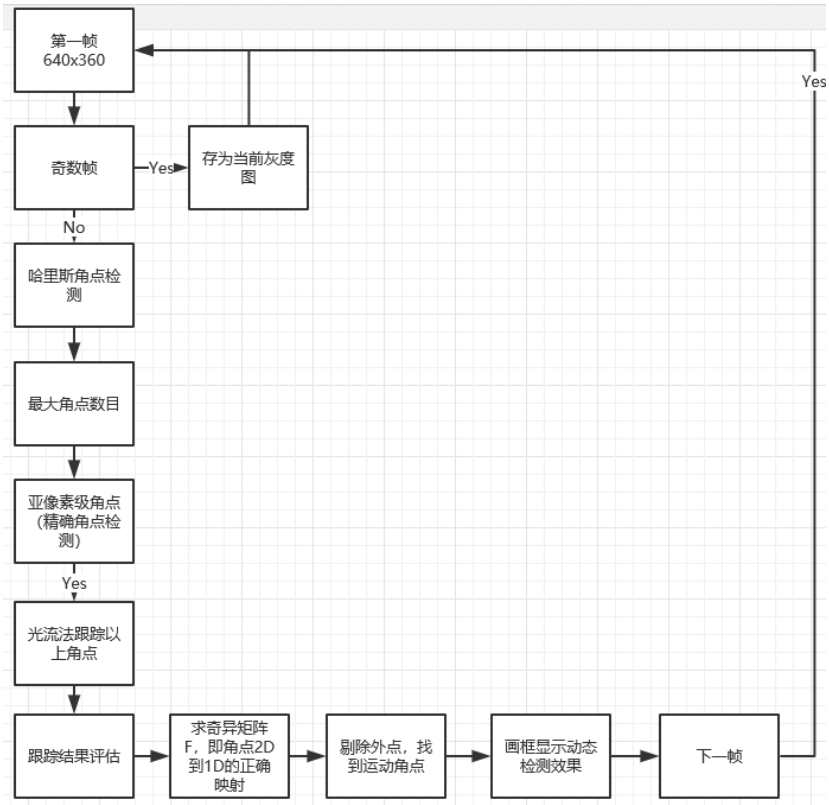
知识点：归一化相关系数法实质

$$NCC(P_1,P_2)=\frac{1}{n-1}\cdot\sum_{i=1}^{n}\frac{(P_1(i)-\mu_1)\cdot(P_2(i)-\mu_2)}{\sigma_1\sigma_2}$$

$NCC(P_1,P_2)$代表图块P_1和P_2的相似度，μ_1和μ_2代表均值，σ_1和σ_2代表标准差。可见模板匹配属于 pixel by pixel 对比的思路。

动态目标检测

1 流程



程序入口：

[1] 抓帧

VideoCapture 获取帧；cap.isOpened()判断是否获取到视频流；成功继续，否则返回；

[2]奇数帧跳过，偶数帧检测

取帧先判断帧数据是否为空 frame.empty()，再利用取余 a%2 != 0 和 continue 判断；

[3]准备工作

设置尺度 scale 的大小为等同于 frame 大小，如 frame.cols*scale=frame.cols;利用 Mat（Size(frame.cols,frame.rows),CV_32SC3）创建 signed char 型 3 通道浮点图 image；将 frame 经过 resize 转换成 image，然后将 image 再由 BGR 格式转换成 gray；

[4]哈里斯角点检测

利用 goodFeaturesToTrack 得到原始角点集 prepoint；清楚 qualityLevel, minDistance, mask, harrisK 设置的意义；

[5]亚像素级角点计算角点的精确位置

利用 cornerSubPix 函数；明白亚像素级定义，结果仍旧保存在 prepoint 中；

[6]光流法跟踪这些角点

这里可以用中值流算法，结果保存在 prepoint 和 nextpoint 中。除了中值流，以上每步算出的结果，角点 size 都是 60；

[7]对跟踪的光流评估有效性

光流法得到一个跟踪结果的状态向量，vector<uchar>型，遍历 state.size();凡是跟踪到，state[i] = 1；否则为 0，据此判断 state[i]=1 时，他们是否越界。如果知道目标大体位置，可以设置边界，过滤不必要的角点（state[i]=0）；其次利用类似 pregray 和 gray 梯度求解，计算像素差值

的和，大于阈值则将 state[i]设置为 0;否则，将该点确认为真正的角点，计数器加 1 且将 prepoint 和 nextpoint 该位置的点保存在 F_prepoint 和 F_nextpoint 中；

[8]求奇异矩阵 F，即角点在 2D 到 1D 映射

定义 Mat F = Mat(3, 3, CV_32FC1); Mat mask = Mat(Size(1, 300), CV_8UC1)；利用 while（flag）语句，迭代求解 F = findFundamentalMat(F_prepoint,F_nextpoint, Mask, FM_RANSAC, 1, 0.99)；

```
for (int i = 0; i < mask.rows; i++) {
    if (mask.at<uchar>(i, 0) == 0){
                        ;
    Else
        根据 F_prepoint 和 F_nextpoint 计算奇异矩阵的系数，利用距离求解公式，计算 dd;
    }
    Flag = false; //dd 是个负值，最终 ppp 会小于 5;
    if(dd >0.1)则什么都不做,小于 0.1 则将 F_prepoint[i]和 F_nextpoint[i]保存到 F2_prepoint 和 F2_nextpint 中。遍历结束后，将 F2_prepoint 和 F2_nextpint 赋值给 F_prepoint 和 F_nextpoint 中。
}
```

[9]运动角点提取

遍历 prepoint, 如果对应的 state[i]!=0, 利用 F 矩阵计算新的 dd，该 dd 是距离极线的距离。如果大于阈值 2，则认为该点前后两帧移动了，于是将其记录为异常的运动角点；否则什么也不做，直到每个点都遍历完；

[10]以角点位置为中心，画图即可

2 API

[1] goodFeaturesToTrack

```
void cv::goodFeaturesToTrack(InputArray _image, OutputArray _corners, int maxCorners,
      double qualityLevel, double minDistance, InputArray _mask, int blockSize, bool
      useHarrisDetector, double harrisK)
```

_image:　　必须是单通道灰度图；

_corners:　vector<Point2f>型，Point2f 原型就是 vector<float>; 同理 Point/Point2i 是 vector<int>,用来存放局部极大值，即角点；

maxCorners:　最大角点数，超过此数字，就排序，选择最强的 maxCorners 个角点；

qualityLevel:和 maxCorners 成反比,如果上面是 200,这里 qualityLevel 建议 0.0X;如果 maxCorners 1000，则 qualitfyLevels 建议 0.00X。因为 Harris 角点计算先检测边缘得到梯度图，完了用 threshold(src, dst, maxCorners*qualityLevel, 0, THRESH_TOZERO).而 THRESH_TOZERO 方法指当 src(i,j) > maxCorners*qualityLevel 时，dst(i,j)=src(i,j)；否则 dst(i,j) =0；故 qualityLevel 取值取决于 maxCorners

minDistance:对于初选出的角点而言，如果在其周围 minDistance 范围内存在其他更强角点，则将此角点删除；值必须>=0;

_mask:　　指定感兴趣区，如不需在整幅图上寻找角点，则用此参数指定 ROI;

blockSize:　计算协方差矩阵时的窗口大小,建议为 3，因为 sobel 算子是 3*3 计算梯度；

useHarrisDetector:　true 即使用哈里斯角点检测，否则用另一个方法

harrisK:　　Harris 角点检测需要的 k 值,一般在 0.04-0.06 之间随意选。

知识点：goodFeaturesToTrack 主要是利用哈里斯角点法检测角点，而哈里斯检测原理见图像处理面试题。

[2] void **cornerSubPix**(InputArray image, InputOutputArray corners, Size winSize, Size zeroZone, TermCriteria criteria);

Image: 单通道灰度图;

Corners: vector<Point2f>型，和上面_corners 一样;

WinSize: 搜索窗口大小，例如如果 winSize=Size(5,5)，则大小为$(5*2+1)(5*2+1)=11*11$的搜索窗口将被使用;

zeroZone：一般默认，用于避免自相关矩阵的奇异性。如果值设为(-1,-1)则表示没有这个区域;

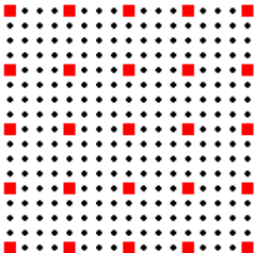
criteria：收敛准则。

知识点:

[1] 亚像素点及精确化像素点的位置，即从一个整数坐标求出一个小数坐标;

[2] 亚像素点求解一般根据最小二乘法迭代，原因在:照相机由光学成像系统，CCD/CMOS 摄像机，信号处理电路，芯片(CPU)组成;CCD 影音质量好，但贵，而 CMOS 质量差，但便宜。一般 CCD 摄像机能形成灰度对比鲜明的边缘。其像素灰度值代表的边缘特征（Sobel 算子）分布符合高斯公式。对高斯公式求 log，可转换为 f（y） = Axˆ2 + Bx + C 形式。由于最多得到两个联立方程，三个未知解，故只能使用迭代方法。求解出 A，B,C 后，才可以对数求逆，才可以求出小数坐标。

[3] 亚像素点求解是根据插值法：比如原来像素 5*5，现在要从整数坐标转为小数坐标，需要将每个像素分为 4 个网格，最后新像素是 16*16，再利用邻近插值，二次插值等方法，求解小数坐标。如下图所示，红色像素代表原来的 5*5，黑色像素代表分割点，一共是 16*16 个小网格。



[4] **void calcOpticalFlowPyrLK**(prevGray, gray, points0, points1, status, err, winSize, 3, termcrit, 0, 0.001);

第一个参数：上一帧灰度图;

第二个参数：当前灰度图;

Points0：vector<Point2f>型，即 2 位单精度浮点型数据组;

points1: 同上，大小类型一致;

status：vector<uchar>型，流跟踪到是 1，否则设置为 0 代表跟踪流失败;

error：对应于上述跟踪结果 1/0 的误差值;

winSize: 搜索窗口大小;

3: 金字塔层数，3 代表 2 层金字塔，5 代表 4 层，以此类推;

Termcrit：收敛准则;

0: 代表操作标志（operation flags），可选参数，即是否把整个 points0 当做原始跟踪点;

最后一个参数 0.001：最小特征根阈值 minEigThreshold，根据跟踪点计算出的特征矩阵的特征根小于此阈值，则抛弃该点。

[5]判断是否越界

Step1: 利用点是否在图像之内;

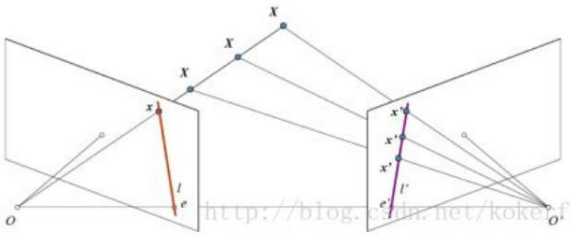
Step2：利用 prevpoint,nextpoint 和某个算子做这些点的类似梯度计算的值，该值要大于 2120 才是我们要找到的哈里斯角点，代码如下；

```
for (int j = 0; j < 9; j++)
    sum_check += abs(prevgray.at<uchar>(y1 + dy[j], x1 + dx[j]) - gray.at<uchar>(y2 + dy[j], x2 + dx[j]));
```

Step3：凡是 sum_check > 2120 阈值，则 harris_corner_point++;

[6]判断运动的角点

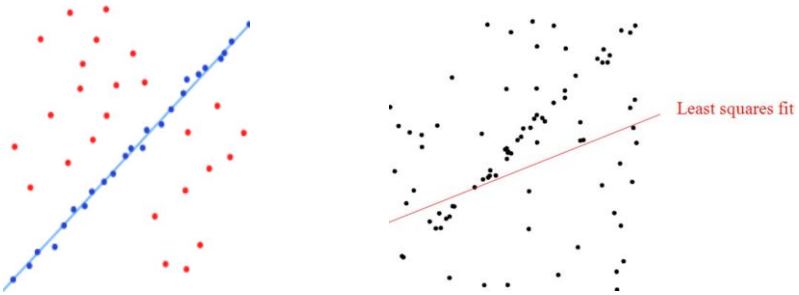
基于上述 4 步，我们得到最终的哈里斯角点。如果要计算焦点中那些是运动的点，则需要借助极线的**对极约束理论**，如下图所示



上图是一个两视图的几何描述，其中O、O'是两个相机的**光心**，两点连线OO'称为**基线**，基线与图像平面的交点$\mathbf{e}$、$\mathbf{e}'$称为**对极点**，其中l、l'分别是图像点$\mathbf{x}'$、$\mathbf{x}$对应的**对极线**。

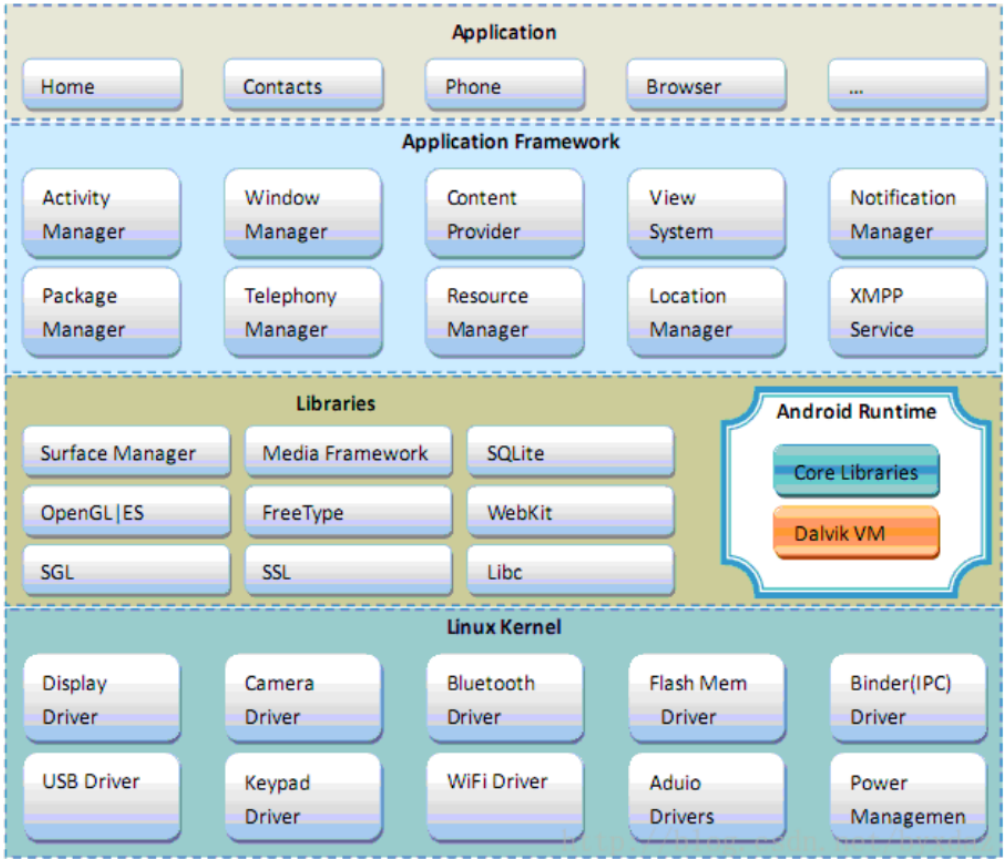
上图的左侧相机的图像平面上的一个点x，反向投影得到射线OX。由于点的深度未知，图像平面上的点x可能是射线上某一深度的3D点X。而射线OX在第二个相机的图像平面上的投影为l'。也就是说，给定一对图像，第一幅图像上的每个点$\mathbf{x}$，在另外一幅图像上存在一条直线l'与之对应。换言之，第二幅图像上与点$\mathbf{x}$对应的点$\mathbf{x}'$必定在线l'上。

例如，利用 findFundamentalMat 基础矩阵判断（点到直线的射影映射关系，和 RANSAC 拟合（findFundamentalMat 函数要设置 RANSAC 参数，系统自动调用模型），把图像间匹配特征的二维搜索转变成沿着极线的一维搜索，排除错误匹配的特征点，最终得到的就是运动特征点。据此特征点坐标为中心，然后在原图像上裁剪出目标位置。最小二乘法不能做，因为最小二乘法只适合线性的拟合和误差较小的拟合，如果点很多，如下图，则建议用 RANSAC 拟合。

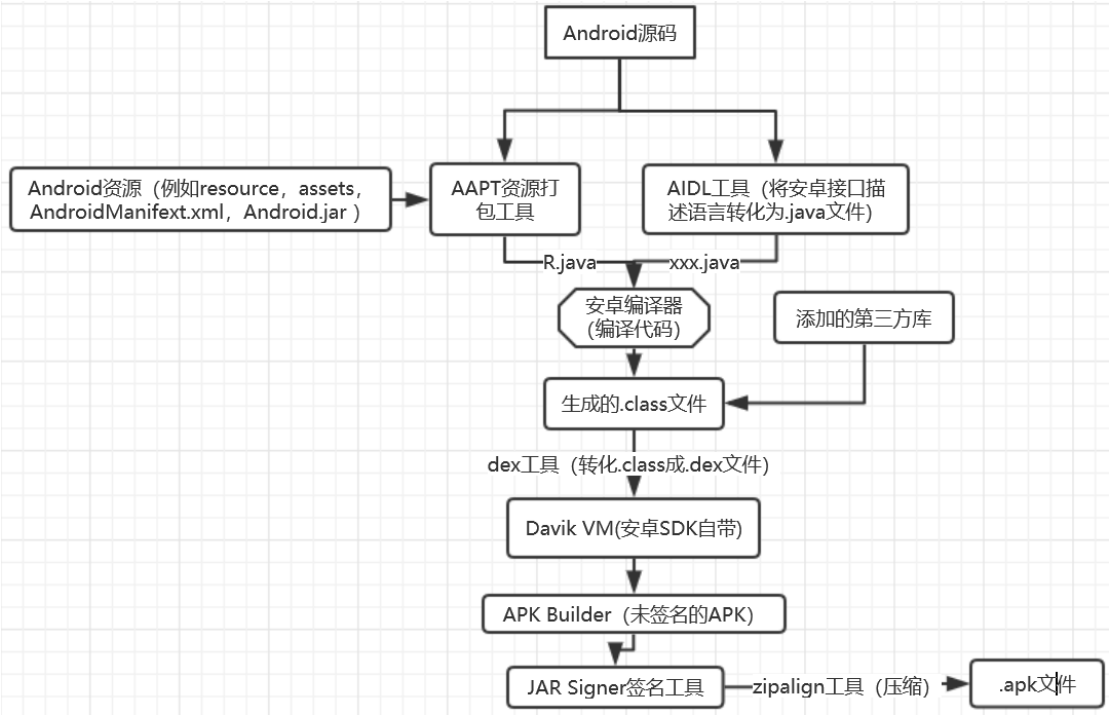


Intel Movidius 使用

1 安卓系统框架



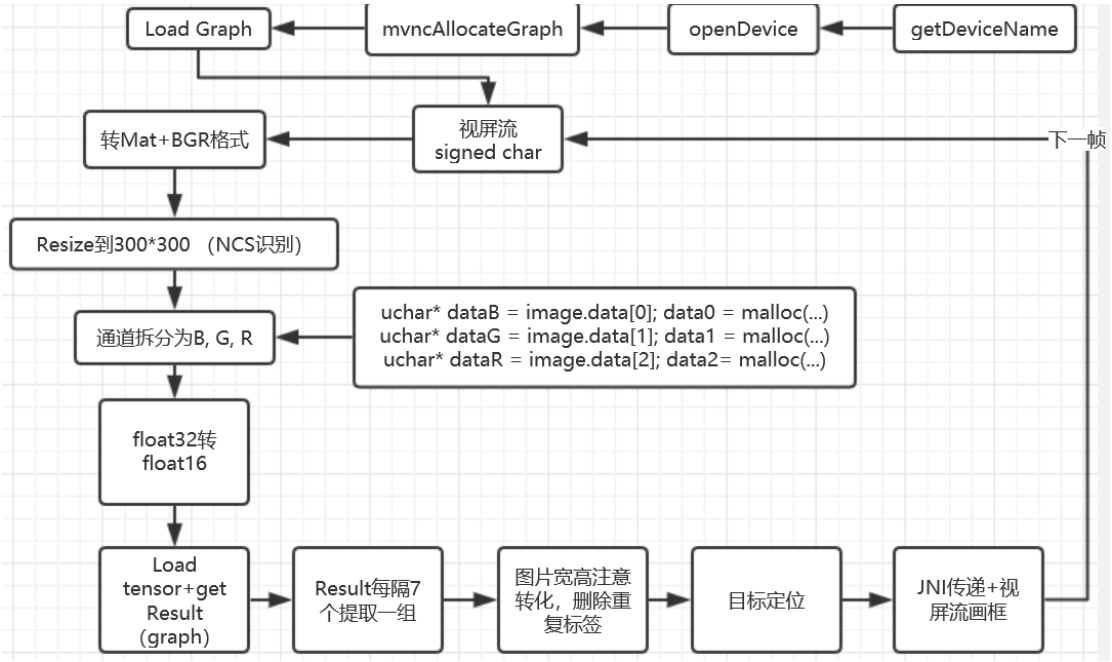
2 安卓生成 APP 流程



3 我在什么地方写代码

Project->app->src->main->com.topotek.addlib->MainActivity.java（系统界面，相机预览，发送广播和窗口命令）

Project->app->src->main->cpp->main.cpp 写代码，代码流程如下



5 adb 命令

答：安卓手机安装 adb 工具，然后添加到 path 路径，用 USB 连接安卓设备。

[1] adb 使用 wifi 连接手机调试

（1）用 USB 连接手机或机芯

（2）查看手机 ip 地址：adb shell ifconfig wlan0；出现 wlan0：192.168.1.3 和 network address: 255.255.255.0 其中 wlan0 即手机的 ip 地址；

（3）cmd 命令行输入 adb tcpip 5555

（4）adb connect 192.168.1.13

注意：有时候 ip 地址总是变化，要么手机被分到的 ip 地址不固定，要么 adb 需要重启，此时用 adb kill-server; adb start-server 即可。

[2]adb install 和 uninstall

（1）安装：adb install -r apk 路径；

（2）卸载：adb uninstall topotek.com.tracker（apk 名）

[3]adb 查看手机 apk 名

（1）adb shell

（2）cd data/data

（3）pm list package

```
package:com.android.soundrecorder
package:com.android.defcontainer
package:com.android.contacts
package:com.android.phone
package:com.google.android.partnersetup
package:com.gradleforandroid.blue.customtype
package:com.jeff.testdemo
package:com.google.android.gsf.login
package:com.android.providers.calendar
package:com.android.inputdevices
```

（4）进入该包：cd com.topotek.demo

（5）显示目录：ls

（6）执行操作，如果查看文件用 cat test.txt 即可显示

[4]adb 查看 Movidius

（1）先 USB 连接没 Movidius 或 TotalRecall

（2）adb shell

（3）cd dev/bus/usb/

（4）ls

主设备是 001，加 OTG 线后的从设备是 002,可看见权限是否打开，如 wrrrww.如果要更改权限，则 chmod 777 之所以是 777 因为

4 – 读取权限；

1 – 运行权限；

2 – 写的权限；

故 7 = 4+2+1 即打开读写运行权限。

6 双 TLD 如何交互

A: JNI 和 CMakelist.txt 要靠自己搞；

B: activity 的 XML 文件安卓工程师设计好；

C: MainActivity Oncreat 函数创建 onCommand()函数还有 UI 按钮组件；

D: private void onCommand(){MainActivity.java 中实现 OnCommand 函数，类似下面

```
public void onCommand(String command) {
        switch (command){
            case ":wKey6":
                sendBroadcast(new Intent("com.topotek.service.data")
                        .putExtra("string", "#TPUD2wTRCAORR"));
                break;
            case ":wKey5":
                sendBroadcast(new Intent("com.topotek.service.data")
                        .putExtra("string", "#TPUD2wTRC1ARR"));
                break;

            //识别
            case ":wKey7":
                Log.i(TAG, "onCommand: --------"+command);
```

```
                JNI.onCommand("A",0);
                break;

            //跟踪
            case ":wKey9":
                Log.i(TAG, "onCommand: --------"+command);
                JNI.onCommand("B",5);
                break;
        }
    }
```

E: onStartPreview()相机开始预览，这里 new 了 JNI 对象: JNI jni = new JNI(this, this);

F: onPreview 相机抓帧，开始 TLD 算法。回传给 native 层可以用 JNI 函数来调用。Native 层收发字符串，用 strcmp((string command,"A")==0);如果相同，则开启 native 层的程序。

7 代码框架

枚举类: mvncStatus retCode; //retCode 即 return code 的意思

第一步：查找设备名，主要是互斥锁保护，以免线程拥挤

```
char devName[Name_Size];
```

retCode = mvncGetDeviceName(0, devName,Name_Size);//第一个参数不能小于 0 否则返回错误参数

内部：pthread_mutex.lock(mutex &mm); 找寻设备代码；pthread_mutex.unlock(mutex &mm);

第二步：打开设备

涉及权限，目的是打开神经计算棒；

第三步：下载 graph，需要提供 graph 在机芯的路径和 graph length，如 sdcard/mvnc/xxx.graph;而 graph 的 length 已经由 intel 公司解析了，长度不得小于文件头长度 264+行长度 227，整体大小不得大于 512*1021*1024；graph length 用的是 unsigned int 型指针；

第四步：allocateGraph，即启动 Movidius 的 Myriad2 处理器，启动内部的 VPU 工作单元，同时讲数据加载到内存中，以便计算识别结果；

第五步：视频流预处理，YUV 数据转 Mat，Mat 转换为指定 300*300 大小，然后拆分通道，总数据 90000 个，对每个通道做

```
double *dpB, *dpG, *dpR;
dpB = (double*)malloc(sizeof(double)*dataLength);
dpG = (double*)malloc(sizeof(double)*dataLength);
dpR = (double*)malloc(sizeof(double)*dataLength);
float *imgfp32 = (float*)malloc(sizeof(float)*lens);
for (int i = 0; i < dataLength; i++) {
        dpB[i] = dataB[i];
        dpG[i] = dataG[i];
        dpR[i] = dataR[i];

        dpB[i] = (dpB[i] - 127.5)*0.007843;  //127.5 是像素均值，0.007843 是 scale 系数
        dpG[i] = (dpG[i] - 127.5)*0.007843;
        dpR[i] = (dpR[i] - 127.5)*0.007843;

        imgfp32[3 * i + 0] = (float)dpB[i]; //把像素值赋给 32 位浮点型数据
```

```
        imgfp32[3 * i + 1] = (float)dpG[i];
        imgfp32[3 * i + 2] = (float)dpR[i];
}
```

第六步：32 位浮点数据转 16 位，涉及内存地址改变，没深入，然后给 loadTensor 函数；

第七步：loadTensor 和 getResult;结果是每 7 位存储一个标签，返回结果是 float 型指针，需要自己根据指针如 p[0]; p[1]; p[2]…p[5]提取，注意宽高是 0-1 之间的比例，所以要换算；

第八步：根据标签和置信度来确定分类，里面有个冒泡排序算法，去除重复的标签，然后画框即可。

8JNI 函数

第一步：CmakeList 配置 native-lib；

第二步：Java 层

[1] MainActivity.java 创建 JNI 对象和插入调用 JNI 的函数

```
Static{
    System.loadLibrary("native-lib");
}
```

在需要调用 jni 的地方写：jni = new JNI(this, this);

[2]在 MainActivity 目录创建 jni 这个包名，包内实现 JNI 类，如

Public static native void onCommand(); //带 native 的是 java 传给 native 层；函数声明在以该类命名的包名+类名的头文件里，也可实现在那里；（getClassName, GetMethodID, CallMethodID）

Public static void resultToJava(); //不带的是 native 层调 java 层；函数实现在 JNI 类中。JNI 层的函数在 jni.h 中，系统默认产生。

第三步：native 层

主要是 C++代码。