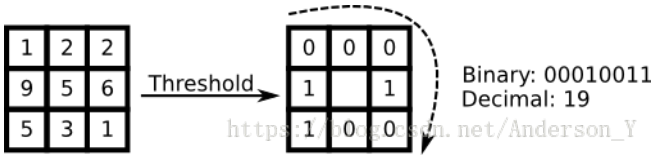
背景减除(Background Segment)

写在前面

1. 高斯背景建模

> 参考：
>
> 博客：运动目标检测_混合高斯背景建模
>
> 高斯背景建模分为单高斯背景建模以及混合高斯背景建模(Gaussian Mixture Model, GMM)。
>
> 高斯背景建模的***核心***点即在于将每个像素点处的颜色值当成一个随机变量，且像素之间的颜色信息相互独立，利用单高斯或混合多高斯函数对该随机变量的分布进行拟合。当一个新的像素点值到来时，将该像素点值与该像素已有的背景模型(单高斯或混合多高斯函数)进行匹配，根据不同的判断准则来判断其是否为背景点。如果判定为背景点，则利用该像素的当前像素值对背景模型进行更新，否则不对背景模型进行任何操作。不同的算法有不同的判断准则以及背景模型的更新方式。

2. LBP特征

> 参考：
>
> 博客：LBP基本原理与特征分析
>
> LBP(Local Binary Pattern)即局部二值模式。这是一种描述图像局部信息纹理特征的算子。像素点LBP特征的提取如下图所示：
>
> 
>
>
> 因此LBP操作可被定义为：
>
> >LBP(xc,yc)=∑p=0P−1(2ps(ip−ic))>(1)(1)>LBP(xc,yc)=∑p=0P−1(2ps(ip−ic))>
>
> 其中 (xc,yc)(xc,yc) 为中心点像素，其亮度值为 icic ，像素点 pp 为其邻域内像素。ss 是一个符号函数：
> >s(x)={>1>0if x≥0others>>(2)(2)>s(x)={>1ifx≥0>0others>>

背景减除算法简介

1. BackgroundSubtractorCNT

> *参考*
>
> github 源码
>
> CNT API
>
> 注：该算法无参考文献，故只根据源码进行简单解释
>
> CNT算法是一种不需要对背景点进行高斯建模处理的方法，它仅仅只使用过去连续N帧内的像素点值的信息以及其他一点额外的信息，因此速度很快，效果也不错。下面根据其源码进行简单解释。
>
> OpenCV的CNT算法有两个版本，一个版本会使用历史记录(useHistory = true)，另一个版本则只考虑最近N帧的数据。
>
> *CNT算法简介*
>
> ***(1) useHistory 参数为 false***

```
void operator()(Vec4i &vec, uchar currColor, uchar prevColor, \
           uchar &fgMaskPixelRef)
  {
     int &stabilityRef = vec[0];
     int &bgImgRef = vec[3];

     if (abs(currColor - prevColor) < threshold)
     {
        ++stabilityRef;


        if (stabilityRef == minPixelStability)
        {
           --stabilityRef;
           bgImgRef = prevColor;
        }
        else
        {
           fgMaskPixelRef = 255;
        }
     }
     else
     {
        stabilityRef = 0;
        fgMaskPixelRef = 255;
     }
  }
```

上面这个函数就是 *useHistory* 参数为 *false* 时的核心函数。代码很简单，就是通过简单的阈值操作判断像素点的稳定性(*stability*)。如果在连续的 *minPixelStability* 帧内都保持稳定，则认为该 像素点是稳定的，否则不稳定。在程序中，稳定的点，即为背景点。

(2) useHistory 参数为 true

```
void operator()(Vec4i &vec, uchar currColor, uchar prevColor,\
          uchar &fgMaskPixelRef)
  {
     int &stabilityRef = vec[0];
     int &historyColorRef = vec[1];
     int &histStabilityRef = vec[2];
     int &bgImgRef = vec[3];

     if (abs(currColor - historyColorRef) < thresholdHistory)
     {
        stabilityRef = 0;
        incrStability(histStabilityRef);
        /************************************************************
         inline void incrStability(int &histStabilityRef)
         {
            // 'maxPixelStability' 参数在创建CNT实例时由程序员指定
            // 默认为 900
            if (histStabilityRef < maxPixelStability)
            {
               ++histStabilityRef;
            }
         }
        ************************************************************/

        if (histStabilityRef <= minPixelStability)
        {
           fgMaskPixelRef = 255;
        }
        else
        {
           bgImgRef = historyColorRef;
        }
     }

     else if (abs(currColor - prevColor) < threshold)
     {
        incrStability(stabilityRef);
        /************************************************************
         inline void decrStability(int &histStabilityRef)
         {
            if (histStabilityRef > 0)
            {
               --histStabilityRef;
            }
         }
        ************************************************************/

        if (stabilityRef > minPixelStability)
        {
           if (stabilityRef >= histStabilityRef)
           {
              historyColorRef = currColor;
              histStabilityRef = stabilityRef;
              bgImgRef = historyColorRef;
           }
           else
           {

              decrStability(histStabilityRef);
              fgMaskPixelRef = 255;
           }
        }
        else
        {
           fgMaskPixelRef = 255;
        }
     }
     else
     {
        stabilityRef = 0;
        decrStability(histStabilityRef);
        fgMaskPixelRef = 255;
     }
  }
```

上面这个函数就是 *useHistory* 参数为 *true* 时的核心函数。程序在运行过程会记录从程序运行到当前时刻为止，稳定时间最长的像素点的灰度值 *historyColorRef* 和它的稳定时间 *histStabilityRef* 。在新的一帧来到时，依然是通过一系列的阈值比较操作来判断像素点的稳定性，从而判断其是否为背景点。

(3) 区别

通过源码，我们不难发现， 两者区别在于是 *useHistory* 参数。 *useHistory* 参数为 *true* 时，程序保存历史记录，并将之作为第一判据，反之程序不会保存历史记录，因而只是简单的将像素点最近一段时间的稳定性作为判据。 *不使用历史记录的好处在于运动物体经过场景(Scene)时，不仅物体当前时刻的位置会被检测标记为前景点，最近一段时间运动经过的地方也会背标记为前景点，换言之，算法可以获得物体最近一段时间的运动轨迹* 。反之， *使用历史记录，可以获得更好的检测效果，检测到的运动区域更准确* 。

运行效果截图

CNT API

```
Ptr<BackgroundSubtractorCNT> cv::bgsegm::createBackgroundSubtractorCNT (
    int minPixelStability = 15,
    bool useHistory = true,
    int maxPixelStability = 15 *60,
    bool isParallel = true  )
```

---

2. BackgroundSubtractorGMG

参考

论文：Visual Tracking of Human Visitors under Variable-Lighting Conditions for a Responsive Audio Art Installation 2012

GMG API

GMG是背景减除算法中的一种。与GMM系列算法不同，该算法并不使用Gaussian 函数对背景进行建模，而是通过像素点的颜色特征对背景进行建模，同时通过贝叶斯公式计算像素点为背景点(前景点)的极大似然估计，得到一张概率图。通过对概率图的阈值操作，来得到前景点和背景点的划分。

GMG算法流程

(1) **Quantization**
对输入的RGB值进行量化，一方面增加对噪声的鲁棒性，另一方面降低存储和计算要求

>>I^ij(k)=floor(q256∗Iij(k))>>(3)(3)>>I^ij(k)=floor(q256∗Iij(k))>>

(2) **Histogram Initialization**
使用输入的前T帧图片对每一个像素点的背景模型进行建模
>>H^ij(T)=1T∗∑r=1Tfij(r)>>(4)(4)>>H^ij(T)=1T∗∑r=1Tfij(r)>>

其中fij(r)fij(r)为Pixel1的颜色值(Pixel Color)
Note :关于这一步，可能有点不准确，但是大概意思就是这个意思。
(3) **Bayesian Inference**
对每一个像素点，根据贝叶斯公式，求它属于背景点的概率，从而得到一张概率图

>>p(B|f)=p(f|B)∗p(B)p(f|B)∗p(B)+p(f|F)∗p(F)>>(5)(5)>>p(B|f)=p(f|B)∗p(B)p(f|B)∗p(B)+p(f|F)∗p(F)>>

其中 p(B)=CONSTANTp(B)=CONSTANT, p(F)=1−p(B)p(F)=1−p(B), p(f|B)=fij(k)T∗H^ij(k)p(f|B)=fij(k)T∗H^ij(k),
p(f|F)=1−p(f|B)p(f|F)=1−p(f|B)
则有 p(B|f)=F(fij(k))p(B|f)=F(fij(k))， 即p(B|f)p(B|f)是fij(k)fij(k)的一个函数。
此时经过这一步操作后，我们得到一张每个像素点为背景点(前景点)的概率图。
(4)**Filtering and Connected Components**
使用形态学操作(开、闭运算)，对得到的概率图进行预处理。使用开(Open)运算可以去掉孤立的高值区域,闭(Close)运算可以去掉孤立的低值区域。执行完上述操作后，再对处理后的图进行一个二值化操作。大于阈值的置为1，反之置为0。其中1代表前景点，0代表背景点。对二值化后的图再进行一次开、闭运算以进一步消除噪声的干扰。

(5) **Updating the Histogram**
当像素点被判定为前景点时，其背景模型不进行更新。仅对被判定为背景点的像素点的背景模型进行更新:

>>Hij(k+1)=(1−a)∗Hij(k)+a∗fij(k)>>(6)(6)>>Hij(k+1)=(1−a)∗Hij(k)+a∗fij(k)>>

参数 *a* 影响背景模型的适应性速率(Adaptation rate)。 *a* 越大，背景模型更新速度越快。

运行效果截图

API

```
Ptr<BackgroundSubtractorGMG> cv::bgsegm::createBackgroundSubtractorGMG (
   int   initializationFrames = 120,
   double    decisionThreshold = 0.8 )
```

---

3. BackgroundSubtractorGSOC

参考

GSOC API

注：该算法无参考论文

运行效果截图

API

```
Ptr<BackgroundSubtractorGSOC> cv::bgsegm::createBackgroundSubtractorGSOC (
  int    mc = LSBP_CAMERA_MOTION_COMPENSATION_NONE,
  int    nSamples = 20,
  float  replaceRate = 0.003f,
  float  propagationRate = 0.01f,
  int    hitsThreshold = 32,
  float  alpha = 0.01f,
  float  beta = 0.0022f,
  float  blinkingSupressionDecay = 0.1f,
  float  blinkingSupressionMultiplier = 0.1f,
  float  noiseRemovalThresholdFacBG = 0.0004f,
  float  noiseRemovalThresholdFacFG = 0.0008f )
```

4. BackgroundSubtractorLSBP

参考

论文：Background Subtraction using Local SVD Binary Pattern 2016

LSBP API

LSBP算法是一种结合LBP特征和SVD(singular value decomposition)的背景减除算法。在参考论文中，作者证明了在假设被检测物体为朗伯体的前提下，LSBP特征具有光照不变性(详细证明过程见参考论文)。

LSBP特征简介*

LSBP(Local SVD Binary Pattern)是在LBP特征基础上的一种改进，通过首先对原始像素值进行SVD操作，再对SVD操作后的图提取LBP特征，得到一个新的对光照具有不变性的特征。LSBP的计算过程如下：

(1) ***对像素进行SVD操作***

将像素及其邻域内的像素值看做一个矩阵，对该矩阵进行SVD操作，得到一系列特征值：

>>B(x,y)=UΣVT>>(7)(7)>>B(x,y)=UΣVT>>

其中 U,VU,V 为正交矩阵，ΣΣ 为对角矩阵，Σ=diag(λ1,λ2,...,λn),and(λ1≥λ2≥...≥λn)Σ=diag(λ1,λ2,...,λn),and(λ1≥λ2≥...≥λn) .
对所有奇异值 λnλn 进行如下操作得到对该像素的一个描述值：

>>g(x,y)=∑j=2Mλj¯>>(8)(8)>>g(x,y)=∑j=2Mλj¯>>

其中 λj¯=λjλ1λj¯=λjλ1 。将得到的 g(x,y)g(x,y) 值作为该像素的新的像素值。
(2) ***提取LSBP特征***

对上一步得到的像素点，进行公式(1) 操作，得到该像素点的 LSBP 特征。

LSBP背景减除算法流程

(1) ***背景建模***

对输入的前 N 帧图像，对每个像素点提取其LSBP特征。将颜色值特征以及LSBP特征分别保存到 BIntindex(x,y)BIntindex(x,y) 以及 BLSBPindex(x,y)BLSBPindex(x,y) 容器中，作为背景模型。

(2) ***提取像素点特征***

对当前帧的每一个像素点，提取其LSBP特征 LSBP(x,y)LSBP(x,y) 和颜色特征 Int(x,y)Int(x,y)

(3) ***匹配***

对每一个像素点，计算其与背景模型中的相匹配的模型个数：

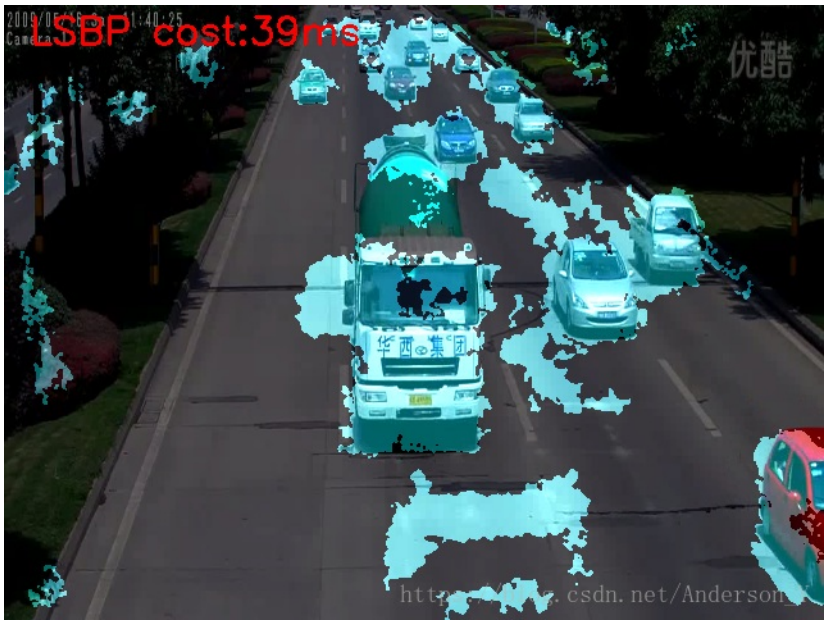
>>match=∑i=1N[L1dist(BInti(x,y),Int(x,y))<R(x,y)\and>>H(BLSBPi(x,y),LSBP(x,y))<H(x,y)]s>>(9)(9)>>match=∑i=1N[L1dist(BInti(x,y),Int(x,y))<R(

(4) ***判断***
根据匹配上的模型个数，判断其是否为背景点：

>>p(x,y)={>>Foreground >>Background if match<MIN_COUNTothers>>>>(10)(10)>>p(x,y)={>>Foregroundifmatch<MIN_COUNT>>Backgr

(4) ***更新背景模型***
匹配判断完成后，根据一定的准则(详细见参考论文)对背景模型进行更新。

运行效果截图



API

```
Ptr<BackgroundSubtractorLSBP> cv::bgsegm::createBackgroundSubtractorLSBP (
    int    mc = LSBP_CAMERA_MOTION_COMPENSATION_NONE,
    int    nSamples = 20,
    int    LSBPRadius = 16,
    float  Tlower = 2.0f,
    float  Tupper = 32.0f,
    float  Tinc = 1.0f,
    float  Tdec = 0.05f,
    float  Rscale = 10.0f,
    float  Rincdec = 0.005f,
    float  noiseRemovalThresholdFacBG = 0.0004f,
    float  noiseRemovalThresholdFacFG = 0.0008f,
    int    LSBPthreshold = 8,
    int    minCount = 2 )
```

---

5. BackgroundSubtractorMOG

参考

论文：An Improved Adaptive Background Mixture Model for Realtime Tracking with Shadow Detection 2001

MOG API

MOG算法简介

MOG算法是一种基于混合高斯背景建模的背景减除算法。算法对每个像素点使用固定数量的高斯函数(Gaussian component)来对其像素值的分布进行建模。通过训练得到每个高斯函数的 (ωi,μi,Σi)(ωi,μi,Σi) 参数。当新的一帧来到时，对场景中的每一个像素计算在 NN 时刻像素值为 \boldxN\boldxN 的概率：

>p(xN)=∑j=1Kωjη(xN;θj)>(11)(11)>p(xN)=∑j=1Kωjη(xN;θj)>

其中 η(xN;θj)η(xN;θj) 为标准正态分布函数：
>η(xN;θj)=η(xN;μj,Σj)>=>e−12(x−μj)TΣ−1j(x−μj)>(2π)D2|Σk|12>(12)(12)>η(xN;θj)=η(xN;μj,Σj)>=>e−12(x−μj)TΣj−1(x−μj)>(2π)D2|Σk|12>

其中 μjμj 和 Σj=σ2jIΣj=σj2I 分别为该高斯函数的均值和方差。对所有 KK 个高斯函数按照 ωj/σjωj/σj 从小到大排序，前B个高斯函数用来对背景进行建模。B按照如下方程选择：
>B=argminb(∑j=1bωj>T)>(13)(13)>B=arg⁡minb⁡(∑j=1bωj>T)>

其中 TT 为一个常数，用于表示场景中背景的最小先验概率(it is the minimum prior probability that the background is in the scene)。对背景建模完成后，对场景中的每一个像素点，如果它与B个高斯函数中的任意一个的距离在 2.5 个标准差以上，则该像素点被认为是前景点(Background subtraction is performed by marking a foreground pixel any pixel that is more than 2.5 standard deviations away from any of the B distributions)。即：
>p={>foreground >background if {∃ (|η(xN;θj)−μj|>2.5σj)|j∈[1,B]}others>>(14)(14)>p={>foregroundif{∃(|η(xN;θj)−μj|>2.5σj)|j∈[1,B]}>backgroundothers

对于第一个匹配上的高斯函数(即与中心的距离在2.5个标准差以内)，采用如下公式对其进行更新：
>>w^N+1k>μ^N+1k>Σ^N+1k>ρ>p^(wk|xN+1)=(1−α)w^Nk+αp^(wk|xN+1)=(1−α)μ^Nk+ρxN+1=(1−α)Σ^Nk+ρ(xN+1−μ^N+1k)(xN+1−μ^N+1k)T=αη(x

如果K个高斯函数都没有匹配上，那么用一个新的高斯函数去替代值最小的那个高斯函数。新高斯函数的均值为当前像素点的均值，方差为一个事先指定的较大值，权重为一个较小的值。

运行效果截图

API

注：在OpenCV实现的MOG算法中，对于前 B 个高斯函数，只要有一个匹配上，则认为该点为背景点。这与论文中稍有不同。

```
dst[x] = (uchar)(kHit < 0 || kHit >= kForeground ? 255 : 0);
```

```
/**
 * @brief: 生成一个MOG算法实例
 * @param history: 算法开始时，用于训练背景模型的帧数
 * @param nmixtures: 高斯函数(Gaussian component)的数量
 * @parma backgroundRatio: 论文中的 T 值。值越大，则相应的 B 值越大，则像素点与高斯函数匹
      配上的可能性越大，即判定为背景点的可能性越大。反之亦然。
 * @param noiseSigma: 新生成高斯函数(Gaussian comoinent)的最小标准差
 */
Ptr<BackgroundSubtractorMOG> cv::bgsegm::createBackgroundSubtractorMOG (
    int   history = 200,
  int     nmixtures = 5,
  double  backgroundRatio = 0.7,
  double  noiseSigma = 0 )
```

6. BackgroundSubtractorMOG2

参考

论文： Efficient adaptive density estimation per image pixel for the task of background subtraction 2004

论文：Improved Adaptive Gaussian Mixture Model for Background Subtraction 2004

MOG2 API

MOG2算法简介

MOG2算法与 MOG 算法基本相同，不同之处在 MOG2 采用可变数量的高斯函数(Gaussian component)。

API

```
Ptr<BackgroundSubtractorMOG2> cv::createBackgroundSubtractorMOG2(
    int   history = 500,
  double  varThreshold = 16,
  bool    detectShadows = true )
```

7. BackgroundSubtractorKNN

参考：

论文： Efficient adaptive density estimation per image pixel for the task of background subtraction 2004

KNN API