总结：光流--LK光流--基于金字塔分层的LK光流--中值流

最近的一个月完成了TLD、CF、Muster等一些算法的学习和整理，由于是在word中整理，不便于再在csdn中编辑，就直接截图发上来了，尽请谅解。（其实还是我自己太懒了，不想再重新编辑一遍了...）如果csdn可以直接发布文档成博客就好了，也希望csdn能够尽快完善这一功能。

本文是对光流算法的理解，从光流--LK光流--基于金字塔分层的LK光流--中值流，可以对光流算法的发展和应用有一个大致的了解。

话不多说，直接上截图。由于理解能力和表达能力有限，如果有错误的地方还请指教~~

光流算法

光流是图像亮度的运动信息描述，这种运动模式指的是由一个观察者（比如眼睛、摄像头等），在一个视角下，一个物体、表面、边缘和背景之间形成的明显移动。它评估了两幅图像之间的变形。

光流计算基于物体移动的光学特性提出了 2 个假设：

①运动物体的灰度在很短的间隔时间内保持不变；

②给定邻域内的速度向量场变化是缓慢的。

假设图像上一个像素点(x,y)，它在时刻 t 的亮度为$I(x,y,t)$，用$u(x,y)$和$v(x,y)$表示该点光流在水平和垂直方向上的**速度分量**。

$$u=\frac{dx}{dt}\quad v=\frac{dy}{dt}$$

在经过时间间隔Δt之后，该点的对应点的亮度变为$I(x+\Delta x,y+\Delta y,t+\Delta t)$。

在运动微小的前提下，利用泰勒公式展开：

$$I(x+\Delta x,y+\Delta y,t+\Delta t)=I(x,y,t)+\frac{\partial I}{\partial x}\Delta x+\frac{\partial I}{\partial y}\Delta y+\frac{\partial I}{\partial t}\Delta t+cons\tan t$$

当Δt足够小，趋近于 0 时：

$$-\frac{\partial I}{\partial t}=\frac{\partial I}{\partial x}\frac{dx}{dt}+\frac{\partial I}{\partial y}\frac{dy}{dt}=\frac{\partial I}{\partial x}u+\frac{\partial I}{\partial y}v$$

$$-I_t=I_x u+I_y v$$

$$-I_t=[I_x\quad I_y][\begin{matrix}u\\v\end{matrix}]$$

这就是基本的光流约束方程。

http://blog.csdn.net/sgfmby1994

但是，基本的光流约束方程的约束只有一个，而需要求出 x，y 方向的速度 u 和 v（两个未知变量）。一个方程两个未知量没法求解，怎么办呢？？LK 光流算法考虑到了像素点的邻域，将问题转变成了计算某些点集的光流，联立多个方程，从而解决了这个问题。

Lucas-Kanade 光流算法

首先，我们从 LK 算法的假设入手，理解它与普通光流算法的区别以及它的特点。

三个假设：

1、**亮度**恒定，图像场景中的目标的像素在帧间运动时外观上保持不变。用于得到光流法基本方程；

2、时间连续或者运动是**小运动**，图像随时间的运动比较缓慢，实际中指的是时间变化相对图像中的运动的比例要足够小。这样灰度才能对位置求偏导（换句话说，小运动情况下我们才能用前后帧之间单位位置变化引起的灰度变化去近似灰度对位置的偏导数），这也是光流法不可或缺的假定；(但是目标运动很快时怎么办呢？？后面会提)

3、邻域内光流一致。一个场景中的同一表面的局部邻域内具有相似的运动，在图像平面上的投影也在邻近区域，且邻近点速度一致（认为邻域内所有像素点的运动是一致的）。这是 Lucas-Kanade 光流法特有的假定。

http://blog.csdn.net/sgfmby1994

说到这里，回顾一下之前提到的光流计算中一个方程两个未知量没法求解的问题，我们发现，第 3 条假设的意义巨大。如果特征点邻域内的所有像素点做相似运动（有着相同的光流），这意味着什么呢？？这意味着我们可以联立 n 个基本的光流方程求取 x，y 方向的速度（n 为特征点邻域内的总点数，包括该特征点）。

$$\begin{cases} I_{1x}u + I_{1y}v = -I_{1t} \\ I_{2x}u + I_{2y}v = -I_{2t} \\ \cdots \\ I_{nx}u + I_{ny}v = -I_{nt} \end{cases}$$

但是，只有两个未知数 u 和 v，却有 n 个方程也是不合理的，这说明其中有的方程是多余的。怎样才能得到最优解呢？

回想一下，这时我们无法做到使每个方程成立，只能使这 n 个方程的偏移量的平方和最小。也就是利用最小二乘法求出这个方程组的最优解。

$$\begin{bmatrix} I_{tx} & I_{ty} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -I_{it} \end{bmatrix}$$

为了简单起见，写成下述形式：

$$A\vec{x} = \vec{z}$$

得到：

$$A^T A\vec{x} = A^T\vec{z}$$
$$\vec{x} = (A^T A)^{-1} A^T\vec{z}$$

所以：

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} I_{ix}^2 & \sum_{i=1}^{n} I_{ix}I_{iy} \\ \sum_{i=1}^{n} I_{ix}I_{iy} & \sum_{i=1}^{n} I_{iy}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_{i=1}^{n} I_{ix}I_t \\ -\sum_{i=1}^{n} I_{iy}I_t \end{bmatrix}$$

即，LK 光流法是一种稀疏光流算法。它假设光流在像素点的局部邻域是一个常数，然后使用最小二乘法对邻域中的所有像素点求解基本的光流方程。从而计算两帧在时间 t 到 $t+\Delta t$ 之间每个像素点位置的移动。

http://blog.csdn.net/sgfmby1994

==========基于金字塔分层的 LK 光流法==========

上面提到了，LK 光流法的第 2 条假定运动是小运动，可是运动快速的时候怎么办？

考虑两帧之间物体的运动位移较大（运动快速）时，算法会出现较大的误差。那么就**希望能减少图像中物体的运动位移**。怎么做呢？缩小图像的尺寸。假设当图像为 400×400 时，物体位移为[16 16]，那么图像缩小为 200×200 时，位移变为[8 8]，缩小为 100*100 时，位移减少到[4 4]。在原图像缩放了很多以后，LK 光流法又变得适用了。

怎样缩小图像的尺寸呢？Bouguet 想到了利用金字塔分层的方式，将原图像逐层分解。简单来说，上层金字塔（低分辨率）中的一个像素可以代表下层的两个像素。这样，利用金字塔的结构，自上而下修正运动量。

具体做法：

一、首先，对每一帧建立一个高斯金字塔，最低分辨率图像在最顶层，原始图片在底层。

http://blog.csdn.net/sgfmby1994

二、如何计算光流呢？从顶层（Lm 层）开始，通过最小化每个点的邻域范围内的匹配误差和，得到顶层图像中每个点的光流。（具体做法见详细过程）

$$\epsilon(\mathbf{d}) = \epsilon(d_x, d_y) = \sum_{x=u_x-\omega_x}^{u_x+\omega_x} \sum_{y=u_y-\omega_y}^{u_y+\omega_y} (I(x,y) - J(x+d_x, y+d_y))^2$$

假设图像的尺寸每次缩放为原来的一半，共缩放了 Lm 层，则第 0 层为原图像。设已知原图的位移为 d，则每层的位移为：

$$d^L = \frac{d}{2^L}$$

三、顶层的光流计算结果（位移情况）反馈到第 Lm-1 层，作为该层初始时的光流值的估计 g。

$$\mathbf{g}^{L-1} = 2(\mathbf{g}^L + \mathbf{d}^L)$$

四、这样沿着金字塔向下反馈，重复估计动作，直到到达金字塔的底层（即原图像）。

$$\mathbf{d} = \mathbf{g}^0 + \mathbf{d}^0$$

（准确值=估计值+残差）注意这个“残差”，是本算法的关键

对于每一层 L，每个点的光流的计算都是基于邻域内所有点的匹配误差和最小化。

$$\epsilon^L(\mathbf{d}^L) = \epsilon^L(d_x^L, d_y^L) = \sum_{x=u_x^L-\omega_x}^{u_x^L+\omega_x} \sum_{y=u_y^L-\omega_y}^{u_y^L+\omega_y} (I^L(x,y) - J^L(x+g_x^L+d_x^L, y+g_y^L+d_y^L))^2$$

这样搜索不仅可以解决大运动目标跟踪，也可以一定程度上解决孔径问题（相同大小的窗口能覆盖大尺度图片上尽量多的角点，而这些角点无法在原始图片上被覆盖）

由于金字塔的缩放减小了物体的位移，也就是减小了光流，所以，作者将最顶层图像中的光流估计值设置为 0。

$$\mathbf{g}^{L_m} = [0 \ \ 0]^T$$

http://blog.csdn.net/sgfmby1994

详细流程

处理目的：

得到从第一帧到第二帧图像中每个点运动的光流情况。即，对于前一帧的图像上一点 (x_0, y_0)，$I(x_0, y_0)$，要在后一帧图像上找到一点 (x_0+dx, y_0+dy) 与之相匹配，即灰度值 $I(x_0, y_0)$ 与 $J(x_0+dx, y_0+dy)$ 最接近。那么向量 d=[dx,dy]就是图像在点 (x_0, y_0) 处的运动位移，也就是像素点 (x_0, y_0) 的光流。

过程：

对连续两帧的图像进行高斯金字塔变换，得到分解后的图像。

http://blog.csdn.net/sgfmby1994

第一帧图像金字塔分解前后对比图

第二帧图像金字塔分解前后对比图

LK 光流法的主过程：

函数：[flowHor flowVer] = pyramidFlow(I1, I2, 5, 3, 3);

[u v] =pyramidFlow(I1, I2, winSize, ITER_NO, PYRE_NO)

作用：经过LK 光流法，得到从 I1 到 I2，每个点的光流情况（即 x 方向和 y 方向）

输入：

I1：第一帧图像

I2：第二帧图像

winSize：邻域大小（integration window）

ITER_NO：每层的迭代次数

PYRE_NO：金字塔层数

输出：

flowHor：最终判定的每个点的光流的 x 方向 水平方向的移动分量

flowVer：最终判定的每个点的光流的 y 方向 垂直方向的移动分量

http://blog.csdn.net/sgfmby1994

详细过程：

1、调整原图像（包括前后两帧图像）的尺寸大小。要保证每层分解得到的尺寸都是整数

因为作者设置了金字塔层数为 3，即要进行 3 层金字塔分解。每次分解，第 L+1 层图像的尺寸是第 L 层图像尺寸的一半。为了保证每层分解得到的图像尺寸都是整数，所以需要在正式处理之前，通过 imresize 对原图像（包括前后两帧图像）的尺寸进行调整。

2、进行原图像（尺寸调整之后）的金字塔分解。形成图像金字塔并初始化其他变量

一幅图像的金字塔是一系列以金字塔形状排列的，越往上层分辨率逐渐降低的图像集合。

金字塔的底部是高分辨率的 raw image，顶部是低分辨率的近似。当向金字塔的上层移动时，图像的尺寸和分辨率都降低。

图像金字塔化包括两个步骤：

①利用低通滤波器（lowpass filter）平滑图像。

平滑图像经常使用的低通滤波器是高斯滤波器，因此用高斯平滑得到的金字塔图像通常也叫作高斯金字塔。

②对平滑图像进行抽样，从而得到一系列尺寸缩小的图像。

图像金字塔化函数：[Apyre, Bpyre, halfWindow] = pyramidInit(A, B_in, PYRE_NO, winSize)

http://blog.csdn.net/sgfmby1994

作用：形成图像金字塔并初始化其他变量

输入：

A：尺寸调整之后的第一帧图像

B_in：尺寸调整之后的第二帧图像

PYRE_NO：金字塔层数

winSize：邻域大小（integration window）

输出：

Apyre：第一帧图像的金字塔

Bpyre：第二帧图像的金字塔

halfWindow：邻域的半径

详细过程：

①利用低通滤波器（lowpass filter）平滑图像。

平滑原理：像素的颜色不仅由自身决定，同时由其周围的像素加权决定，客观上减小与周围像素的差异，就能起到平滑图像的作用。

周围像素的权重满足距离中心越近，权重越大的规律。也就是说，从理论上满足高斯分布。

$$G(x,y)=\frac{1}{2\pi\delta^2}\exp(-\frac{(x-\mu_x)^2+(y-\mu_y)^2}{2\delta^2})$$

（μ 为均值（峰值对应位置））

因此，作者生成了一个标准差为 1，大小为 3×3 的高斯模板矩阵，并归一化，作为低通滤波器。

```
G = fspecial('gaussian',[3 3],1);
```

这样，得到这个低通滤波器 G 为：

G <3x3 double>

	1	2	3
1	0.0751	0.1238	0.0751
2	0.1238	0.2042	0.1238
3	0.0751	0.1238	0.0751

http://blog.csdn.net/sgfmby1994

高斯滤波是将输入数组的每一个像素点与高斯内核卷积，将卷积和当作输出像素值。

```
Apyre{1} = conv2( A, G, 'same' );
Bpyre{1} = conv2( B_in, G, 'same' );
```

下图为第一帧和第二帧平滑前后的第一层图像对比，可以看出，平滑后的图像相比之前变得模糊了。

平滑前　　平滑后

http://blog.csdn.net/sgfmby1994

②对平滑图像利用 impyramid 进行抽样，生成图像金字塔，从而得到一系列尺寸缩小的图像。

第 3 层：  

第 2 层：  

第 1 层：  

第 0 层：  

第一帧的图像金字塔　　第二帧的图像金字塔

http://blog.csdn.net/sgfmby1994

3、金字塔跟踪

总体流程：

首先，从顶层开始计算金字塔最顶层图像上的光流。然后，根据最顶层（Lm-1）光流的计算结果估计次顶层光流的初始值，再计算次顶层图像上光流的精确值。最后，根据次上层光流的计算结果估计下一层（Lm-2）光流的初始值，计算其精确值后再反馈到下一层，直至计算出最底层的原始图像的光流。

第 Lm 层：

对前后两帧的本层图像添加额外像素圈，什么是额外像素圈呢？当需要计算的图像中的点接近边缘时，它的邻域内的点有可能会超出边界。如果强制所有的邻域窗口都在图像以内，那么图像中会存在宽度等于邻域窗口半径的 forbidden band。也就是说，这个 forbidden band 内的点无法计算光流。为了解决这些接近边缘的点的邻域超出图像边界，从而无法正常计算它们的光流的问题，也就是说即使点的邻域窗口超出图像范围时仍然可以正常计算。作者在每层图像中光流计算之前，在图像周围添加了额外的像素圈，将像素圈内的像素值填充为图像的真实边界值。这样在计算边缘点的邻域时，相当于只计算该点邻域的有效部分。如图：

顶层原图　　添加额外像素圈之后的顶层图像

对比（红框内的原图，红框外为额外添加的像素圈）

http://blog.csdn.net/sgfmby1994

在顶层，初始化顶层图像中的光流估计值为 0（尺寸很小，所以近似为 0）。由于金字塔的缩放减小了光流值，最高层的光流估计值可以设为 0

$$\mathbf{g}^{L_m} = [g_x^{L_m} \quad g_x^{L_m}]^T = [0 \quad 0]^T$$

论文中重新定义了一个速度向量，即光流情况。

$$\overline{\nu} = [\nu_x \quad \nu_y]^T = \mathbf{d}^{L}$$

这时，邻域内所有像素点的匹配误差和记为：

$$\varepsilon(\overline{\nu}) = \varepsilon(\nu_x, \nu_y) = \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} (A(x,y) - B(x+\nu_x, y+\nu_y))^2$$

（其中，A 和 B 为坐标处的灰度值）

对每层的光流计算就是利用最小二乘法，求邻域内匹配误差和的导数，在最优解处，导数为 0，即匹配误差和最小。此时两帧图像对应点之间相似度最高。

$$\left.\frac{\partial \varepsilon(\overline{\nu})}{\partial \overline{\nu}}\right|_{\overline{\nu}=\overline{\nu}_{\text{opt}}} = [0 \quad 0]$$

匹配误差和的导数为：

$$\frac{\partial \varepsilon(\overline{\nu})}{\partial \overline{\nu}} = -2 \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} (A(x,y) - B(x+\nu_x, y+\nu_y)) \cdot \begin{bmatrix} \frac{\partial B}{\partial x} & \frac{\partial B}{\partial y} \end{bmatrix}$$

利用泰勒公式的前提是每次的位移变化是很微小的。为什么可以这样用呢？？这个问题将在下一页详细解答。对 $B(x+v_x, y+v_y)$ 进行展开：

$$B(x+v_x, y+v_y) \approx B(x,y) + \begin{bmatrix} \frac{\partial B}{\partial x} & \frac{\partial B}{\partial y} \end{bmatrix} \overline{v}$$

http://blog.csdn.net/sgfmby1994

以此，导数公式变为：

$$\frac{\partial \varepsilon(\overline{\nu})}{\partial \overline{\nu}} \approx -2 \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \left(A(x,y) - B(x,y) - \begin{bmatrix} \frac{\partial B}{\partial x} & \frac{\partial B}{\partial y} \end{bmatrix} \overline{\nu}\right) \cdot \begin{bmatrix} \frac{\partial B}{\partial x} & \frac{\partial B}{\partial y} \end{bmatrix}$$

其实，我们可以发现：（与第 2 页的红框内的公式对比）

$$-I_t = A(x,y) - B(x,y) = \delta I$$

$$\nabla I = \begin{bmatrix} I_x \\ I_y \end{bmatrix} \doteq \begin{bmatrix} \frac{\partial B}{\partial x} & \frac{\partial B}{\partial y} \end{bmatrix}^T$$

代入原公式，可以得到：

$$\frac{1}{2} \frac{\partial \varepsilon(\overline{\nu})}{\partial \overline{\nu}} \approx \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} (\nabla I^T \overline{\nu} - \delta I) \nabla I^T$$

即：

$$\frac{1}{2} \left[\frac{\partial \varepsilon(\overline{\nu})}{\partial \overline{\nu}}\right]^T \approx \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \left(\begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \overline{\nu} - \begin{bmatrix} \delta I I_x \\ \delta I I_y \end{bmatrix} \right)$$

定义：

$$G \doteq \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \qquad \overline{b} \doteq \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \begin{bmatrix} \delta I I_x \\ \delta I I_y \end{bmatrix}$$

所以：

http://blog.csdn.net/sgfmby1994

$$\frac{1}{2}\left[\frac{\partial \varepsilon(\overline{v})}{\partial \overline{v}}\right]^{T} \approx G\overline{v} - \overline{b}$$

由于：

$$\left.\frac{\partial \varepsilon(\overline{v})}{\partial \overline{v}}\right|_{\overline{v}=\overline{v}_{\text{opt}}} = [0 \;\; 0]$$

所以光流向量的最优解为：

$$\overline{v}_{\text{opt}} = G^{-1}\overline{b}$$

这个公式很关键！！对于每层图像，其光流都是依据这个公式计算得到的。

观察 G 和 $\overline{b}$ 的组成，很容易求得 δI，即是两帧同层图像灰度值的差值。I_x 和 I_y 分别是图像在该点的梯度的 x 方向分量和 y 方向分量。那么，某个点的梯度如何计算呢？？（中心差分方法）请看下面的解释。

```
[Ix Iy] = gradient( B_ref );
%gradient()是求数值梯度函数的命令。
%[Fx,Fy]=gradient(x)，
%其中Fx为其水平方向上的梯度，Fy为其垂直方向上的梯度，
%Fx的第一列元素为原矩阵第二列与第一列元素之差，
%Fx的第二列元素为原矩阵第三列与第一列元素之差除以2，
%Fx的第三列元素为原矩阵第四列与第二列元素之差除以2，
%以此类推：Fx(i,j)=(F(i,j+1)-F(i,j-1))/2。最后一列则为最后两列之差。
%同理，可以得到Fy。
```

从而，根据 I_x 和 I_y，得到空间梯度矩阵 G 和 $\overline{b}$。以此得到第 Lm 层图像的最佳光流 d^{Lm}。

这里需要注意：由于之前在每层图像上添加了额外的像素圈，这部分并不会产生光流，需要把这个额外的像素圈去除掉，只得到有效部分，即本层图像中每个点的光流情况（水平方向和垂直方向的位移）。

us <34x44 double>

	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	1.4002	2.4178	2.0913	1.6667	1.4933	1.3416	1.0801	0.7576
4	0	0	1.4714	2.4175	1.7455	1.5038	1.4456	1.2499	1.0683	0.8962
5	0	0	0.8135	1.9191	1.2298	1.1596	1.2128	1.0942	0.9050	0.9248
6	0	0	0.7284	1.6508	0.9379	0.8767	1.0513	0.9819	0.8353	0.9551
7	0	0	0.7831	1.5547	0.8030	0.7411	1.0213	0.9366	0.8259	0.9232
8	0	0	0.7772	1.5866	0.7997	0.7658	1.0544	0.9108	0.8203	0.9079
9	0	0	0.6479	1.5636	0.9361	0.9262	1.0968	0.9348	0.8620	0.9099
10	0	0	0.9855	1.5415	1.0303	0.9073	1.0860	0.9003	0.9303	0.9379
11	0	0	1.1638	1.4866	1.0529	0.9563	1.1320	0.9463	0.9932	0.9396
12	0	0	1.2269	1.4512	1.1471	1.0619	1.1480	0.9576	0.9970	0.9408
13	0	0	1.5379	1.3841	1.1913	1.1223	1.1104	0.9592	1.0718	1.0236
14	0	0	1.7697	1.3672	1.2404	1.2808	1.1486	1.1005	1.2082	1.1104
15	0	0	0.8887	1.2934	1.3748	1.3524	1.1589	1.0545	1.0028	1.0739
16	0	0	0.3474	0.9436	1.3765	1.2261	0.9620	0.6021	0.5115	0.7738
17	0	0	0.2731	0.9089	1.4841	1.1704	0.7596	0.2614	0.1375	0.2074
18	0	0	0.2212	0.8965	1.5640	1.1388	0.3440	-0.4681	-0.8400	-0.9643

去除前

us <30x40 double>

	1	2	3	4	5	6	7	8	9
1	1.4002	2.4178	2.0913	1.6667	1.4933	1.3416	1.0801	0.7576	0.9761
2	1.4714	2.4175	1.7455	1.5038	1.4456	1.2499	1.0683	0.8962	0.9963
3	0.8135	1.9191	1.2298	1.1596	1.2128	1.0942	0.9050	0.9248	0.9954
4	0.7284	1.6508	0.9379	0.8767	1.0513	0.9819	0.8353	0.9551	1.0073
5	0.7831	1.5547	0.8030	0.7411	1.0213	0.9366	0.8259	0.9232	0.9601
6	0.7772	1.5866	0.7997	0.7658	1.0544	0.9108	0.8203	0.9079	0.9376
7	0.6479	1.5636	0.9361	0.9262	1.0968	0.9348	0.8620	0.9099	0.9388
8	0.9855	1.5415	1.0303	0.9073	1.0860	0.9003	0.9303	0.9379	0.9539
9	1.1638	1.4866	1.0529	0.9563	1.1320	0.9463	0.9932	0.9396	0.9785
10	1.2269	1.4512	1.1471	1.0619	1.1480	0.9576	0.9970	0.9408	0.9278
11	1.5379	1.3841	1.1913	1.1223	1.1104	0.9592	1.0718	1.0236	1.0013
12	1.7697	1.3672	1.2404	1.2808	1.1486	1.1005	1.2082	1.1104	1.1126
13	0.8887	1.2934	1.3748	1.3524	1.1589	1.0545	1.0028	1.0739	1.0697
14	0.3474	0.9436	1.3765	1.2261	0.9620	0.6021	0.5115	0.7738	0.6310
15	0.2731	0.9089	1.4841	1.1704	0.7596	0.2614	0.1375	0.2074	0.0117
16	0.2212	0.8965	1.5640	1.1388	0.3440	-0.4681	-0.8400	-0.9643	-1.0842

去除后

http://blog.csdn.net/sgfmby1994

反馈到下一层：

设置下一层光流的估计值。

$$\mathbf{g}^{L-1} = 2\left(\mathbf{g}^{\mathbf{L}} + \mathbf{d}^{L}\right)$$

（因为上层金字塔（低分辨率）中的一个像素可以代表下层的两个像素，所以在向上一层反馈时，光流向量需要扩大 2 倍）

注意!!! 在每层图像上，都要依据光流估计值预先平移邻域窗口，这样操作之后计算的残差位移向量 d 就非常小，就可以通过标准 LK 算法进行计算了。这也就是为什么 $B(x+v_x, y+v_y)$ 可以利用泰勒公式展开的原因，v_x 和 v_y 是很微小的。（之前看了很长时间，没有明白设置这个光流估计值有什么用。这个是核心啊!! 如果不在每层图像计算光流之前，先根据光流估计值预先平移邻域窗口，就不能保证每次迭代的残差位移向量 d 是微小的，进而就无法利用泰勒公式。这正是整个算法的核心，这一步很重要!!!）

http://blog.csdn.net/sgfmby1994

对于第 Lm-1 层的光流估计值：

$$g^{Lm-1} = 2(g^{Lm} + d^{Lm}) = 2(0 + d^{Lm}) = 2d^{Lm}$$

对于最终的原图像，光流值为：

$$\mathbf{d} = \mathbf{g}^0 + \mathbf{d}^0$$

可以看出，其实最终的光流值就是所有层的分段光流 d 的叠加。

$$\mathbf{d} = \sum_{L=0}^{L_m} 2^L \mathbf{d}^L$$

这时候，我们可以思考一下使用金字塔图像计算光流的好处。对于每一层的光流都会保持很小，但是最终计算来的光流可以进行累积，所以利用相对小的邻域窗口就可以处理较大的像素位移。

结果：

flowHor：最终判定的每个点的光流的 x 方向 水平方向的移动分量

flowVer：最终判定的每个点的光流的 y 方向 垂直方向的移动分量

flowHor <240x320 double>

	1	2	3	4	5	6	7	8	9	10
1	10.5654	10.5199	10.4290	9.3819	7.3787	6.1608	5.7282	5.4804	5.4174	5.162
2	10.5782	10.4910	10.3166	9.2182	7.1958	5.9644	5.5239	5.2732	5.2122	4.971
3	10.6038	10.4331	10.0917	8.8907	6.8299	5.5715	5.1154	4.8588	4.8018	4.589
4	10.5796	10.3829	9.9897	8.6942	6.4965	5.2245	4.8780	4.6432	4.5200	4.335
5	10.5056	10.3405	10.0105	8.6288	6.1956	4.9233	4.8119	4.6264	4.3670	4.209
6	10.3914	10.3379	10.2310	8.9524	6.5022	5.4308	5.7380	5.7242	5.3894	5.192
7	10.2370	10.3751	10.6513	9.6651	7.4165	6.7470	7.6565	7.9366	7.5871	7.284
8	10.1372	10.3125	10.6630	9.7666	7.6233	7.2171	8.5481	9.0743	8.7957	8.447
9	10.0920	10.1500	10.2662	9.2570	7.1225	6.8411	8.4128	9.1374	9.0150	8.679
10	10.1547	10.1094	10.0188	8.8797	6.6921	6.4392	8.1210	8.9547	8.9405	8.629
11	10.3253	10.1905	9.9210	8.6349	6.3322	6.0114	7.6726	8.5262	8.5721	8.298
12	10.5971	10.4733	10.2258	8.9398	6.6154	6.2319	7.7891	8.5905	8.6362	8.387
13	10.9702	10.9578	10.9331	9.7945	7.5419	7.1005	8.4703	9.1478	9.1329	8.896
14	11.3645	11.4941	11.7532	10.7428	8.4629	7.8252	8.8298	9.3584	9.4112	9.364
15	11.7802	12.0822	12.6861	11.7849	9.3785	8.4060	8.8673	9.2224	9.4711	9.790
16	11.9438	12.4912	13.5860	12.8569	10.3036	9.0650	9.1412	9.3551	9.7067	10.252
17	11.8552	12.7211	14.4531	13.9588	11.2383	9.8025	9.6513	9.7566	10.1181	10.750

flowVer <240x320 double>

	1	2	3	4	5	6	7	8	9	10
1	-0.6087	-0.5312	-0.3763	-0.2343	-0.1053	0.0586	0.2574	0.6524	1.2435	1.7197
2	-0.3629	-0.3039	-0.1859	-0.0203	0.1929	0.4873	0.8628	1.3503	1.9497	2.4219
3	0.1285	0.1506	0.1949	0.4078	0.7895	1.3447	2.0735	2.7460	3.3623	3.8264
4	0.7571	0.7314	0.6800	0.9107	1.4233	2.0838	2.8922	3.6010	4.2104	4.5202
5	1.5227	1.4383	1.2695	1.4883	2.0946	2.7047	3.3188	3.9153	4.4942	4.5031
6	2.1001	1.9498	1.6493	1.7898	2.3715	2.9386	3.4913	4.0110	4.4977	4.4143
7	2.4893	2.2660	1.8193	1.8153	2.2540	2.7855	3.4097	3.8882	4.2209	4.2537
8	2.7961	2.5258	1.9853	1.8735	2.1905	2.6446	3.2360	3.6389	3.8533	3.8679
9	3.0204	2.7294	2.1475	1.9646	2.1807	2.5160	2.9702	3.2632	3.3948	3.2569
10	3.2398	2.9476	2.3633	2.1622	2.3443	2.6139	2.9709	3.1579	3.1748	2.9009
11	3.4542	3.1804	2.6328	2.4664	2.6812	2.9384	3.2381	3.3230	3.1932	2.7998
12	3.2068	3.0012	2.5900	2.4961	2.7195	2.9500	3.1875	3.2072	3.0090	2.5873
13	2.4977	2.4101	2.2349	2.2513	2.4592	2.6485	2.8192	2.8104	2.6221	2.2634
14	2.2101	2.2018	2.1852	2.2465	2.3858	2.5035	2.5996	2.5395	2.3232	1.9687
15	2.3442	2.3764	2.4409	2.4818	2.4992	2.5149	2.5288	2.3946	2.1120	1.7032
16	2.6141	2.7225	2.9392	2.9789	2.8417	2.7702	2.7644	2.6137	2.3183	1.8887
17	3.0199	3.2399	3.6801	3.7378	3.4133	3.2694	3.3061	3.1970	2.9419	2.5252
18	3.1655	3.6307	4.5611	4.6868	4.0077	3.7113	3.7976	3.7480	3.5625	3.1779

这里注意，这个位移指的是第二帧中的像素点移动多少会和第一帧最相似，所以可以根据 flowHor 和 flowVer，对第二帧图像中的像素点进行移动，得到 warped 的第二帧，原则上，这个 warped 之后的第二帧应该和第一帧图像很相似。

第一帧原图

warped 之后的第二帧

4、每层图像中的迭代过程

迭代目的：为了得到更精确的结果

为了得到更精确的结果，在金字塔的每一层，需要迭代多次来求得精确的最优解。由于每一层的迭代过程是相同的，所以我们就描述从第 L+1 层到第 L 层的迭代过程。强化目标：计算出使邻域内匹配误差和最小的光流 d^L 。

假设已经从第 L+1 层图像计算得到了第 L 层光流的初始估计值 $\mathbf{g}^L = [g_x^L \; g_y^L]^T$ 。为了计算第 L 层的

光流，需要使得图像间的匹配误差函数达到最小。在这之前，别忘了需要依据上一次的估计值，平移邻域窗口，使得每次得到的残差光流都很微小，这才可以使用标准的 LK 算法（因为 LK 算法的使用假设是运动是微小的（根源是因为需要使用泰勒展开）），每次计算出的残差光流可递推下一次迭代的光流估计值。迭代过程将一直进行，直到计算出的残差光流小于给定的阈值或者达到了最大的迭代次数。假设 K 次达到收敛，则本层图像的最终残差光流向量是本层所有迭代得到的残差光流向量之和。

跟踪中遇到的问题：

前一帧中的特征点并不都会在后一帧中被跟踪成功，有些点会跟丢，跟丢的原因有：

1、特征点已超出图像。这种情况自然而然就可认为该点跟丢了

2、虽然通过最小化邻域范围内的匹配误差和（SSD）找到了下一帧中的对应点，但是这个匹配误差和太大了（大于一个阈值）。

而且，即使跟踪到了，也不能说明跟踪到的点都是两帧之间正确匹配的点，而错误匹配的点将影响下一帧中目标位置的判断，所以还需要一些方法（如 RANSAC）来剔除错误匹配的点。http://blog.csdn.net/sgfmby1994

==========特征点的选择==========

LK 光流跟踪可以完成对所有像素点的光流计算，但会产生很大的时间开销（两帧之间计算光流就长达 2s），所以我们需要在图像中选取一些点作为特征点，然后利用基于金字塔的 LK 光流法跟踪这些点在下一帧中的位置。怎样从那么多像素点选取一些点作为特征点呢，选取的依据是什么呢？下面介绍几种特征点选取的方法。

一、在《Good features to track》这篇论文里提到了利用梯度矩阵的特征值筛选像素点的方法。

因为 LK 光流法的核心是计算每次的光流向量，其中要求矩阵 G 可逆。

$$\overline{v}_{\text{opt}} = G^{-1}\,\overline{b}$$

$$G \doteq \sum_{x=p_x-\omega_x}^{p_x+\omega_x}\sum_{y=p_y-\omega_y}^{p_y+\omega_y}\begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

所以我们先来回顾一下矩阵可逆的条件：

A 可逆的充要条件是|A|≠0

<=> Ax=0 只有零解

<=> Ax=b 总是有解

<=> A 的列向量组线性无关

<=> A 的行向量组线性无关

<=> A 的特征值都不等于零

也就是要求矩阵 G 的最小特征值要足够大，满足这个要求的像素点才易于跟踪。

因此，可按照以下步骤选取特征点：

1、计算图像 I 中每个像素的矩阵 G，并记录每个 G 的最小特征值 λ_m。

2、寻找整幅图像每个 G 的最小特征值 λ_m 中的最大值 $\lambda_{\max}$。

3、保留最小特征值 λ_m 大于给定阈值的 G 所对应的像素点。阈值通常设置为 $5\%\lambda_m \sim 10\%\lambda_m$。

4、从上一步得到的像素点中保留特征值局部最大的像素点。（如果一个像素点的 λ_m 大于其周围 3×3 邻域内其他像素点的特征值 λ_m，则保留这个像素点）http://blog.csdn.net/sgfmby1994

5、剔除上一步保留的像素点中分布密集的一些像素点，确保任何像素点之间的距离都大于给定的阈值（通常设置 5~10 像素）

这样，仍然保留的像素点即为选取的特征点，用于帧间跟踪。

二、在《Machine learning for high-speed corner detection》这篇论文里提到了检测角点，将检测到的角点作为特征点。

三、这两种方法都是基于从单幅图像提取信息，在《Forward-Backward Error: Automatic Detection of Tracking Failures》这篇论文中提到了基于 FB error 进行特征点提取的方法。这是考虑到整个序列的信息的提取方法。

在整个序列上跟踪每个像素点，利用 FB error 评估每个像素点的轨迹，这样构建 Error Map，用每个像素点的 FB error 作为 Error Map 的值。FB error 小于阈值的点认为是可靠的像素点，即特征点。

http://blog.csdn.net/sgfmby1994

==========基于中值流（Median Flow）的目标跟踪==========

既然提到了 FB error，就要说到 FB error 的原作者在 FB error 的基础上提出的中值流（Median Flow）目标跟踪算法。TLD 算法的跟踪模块借鉴了这个中值流算法用于上一帧目标 bb 到当前帧目标 bb 的预测。中值流算法的流程如下：

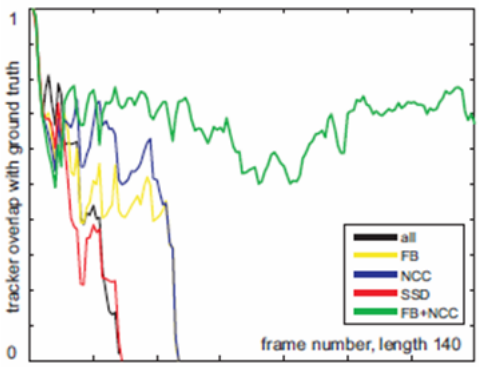
①初始化第 t 帧中的目标 bb，在 bb 内均匀取点用于跟踪。TLD 的作者在目标 bb 内均匀选取了 100 个像素点作为跟踪点。

②利用 LK 光流法确定第 t 帧与第 t+1 帧之间的光流情况。用预测到的第 t+1 帧中这些点的位置，反向利用 LK 光流法得到这些点的后向跟踪轨迹。这样，每个跟踪点都会在第一帧中由前向和后向得到两个位置，这两个位置的欧式距离就作为每个点的 FB error。

③计算每个跟踪点在两帧之间的匹配度（即计算 ncc 的值）。

④利用所有的跟踪点预测bb的变化有失妥当，因为有的跟踪点可能并不对bb的变化起作用，甚至会影响bb的预测（见下图黑线所示）。所以需要筛选得到一部分可靠的点。这也就是中值流算法的核心部分。

将这些跟踪点在相邻两帧之间的FB_error和ncc值进行排序，得到中值，利用中值，得到同时小于FB_error中值与ncc中值的特征点(分别通过50%过滤跟踪点)，这样保留不到50%的点作为跟踪成功的点。为啥不用匹配误差和（SSD）用于筛选呢？？因为FB_error的作者通过实验看出利用SSD判定跟踪点是否可靠并不是很靠谱。



⑤利用剩下的这些点来预测上一帧的目标bb在当前帧的位置和大小。

位移估计：用保留的这些点的 x，y 位移的中值作为目标 bb 的位移。

尺度估计：计算上一帧中保留的这些点之间的欧氏距离 d1，当前帧中保留的这些点之间的欧氏距离 d2。并以此得到伸缩比的中值 s。s = median(d2./d1) 这个伸缩比的中值作为目标 bb 尺度伸缩值。

相关推荐

关于我们 招贤纳士 广告服务 开发助手 400-660-0108 kefu@csdn.net 在线客服 工作时间 8:30-22:00

公安备案号11010502030143 京ICP备19004658号 京网文〔2020〕1039-165号 经营性网站备案信息 北京互联网违法和不良信息举报中心 网络110报警服务 中国互联网举报中心 家长监护 Chrome商店下载 ©1999-2021北京创新乐知网络技术有限公司 版权与免责声明 版权申诉 出版物许可证 营业执照