1. 原始DIS稠密光流思路

==========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。

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。

$$\begin{bmatrix} I_{ix} & I_{iy} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -I_{it} \end{bmatrix}$$

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。

==========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。

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。

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

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。

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

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。

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

2. OpenCV中DIS源码具体优化策略

（1）初始化。

- 考虑到视频是连续的，当前帧计算的一些特征值，可以作为下一帧的前一帧特征值，无需重复计算；
- **删除一些中间变量的重复初始化；**
- **前一帧的光流作为当前帧光流计算的初始值，可以减少搜索范围。**

（2）光流稠密化时，除了最精细层，其余层不考虑块重叠，减少插值运算。

（3）较粗层光流转化为下一层光流初始值时，采用最近邻插值算法。

（4）梯度下降算法中动态化学习率，加快收敛速度。原始的学习率始终为1，我们优化为动态学习率，迭代初期学习率大于1，随着迭代次数的增加，学习率逐渐下调。

以上的思路主要是参考谷歌google pixel2 优化稀疏光流算法的思路：

（1）运动估计的优化思路（Google Pixel 2）

- **在KLT金字塔稀疏光流追踪过程中，在更粗的金字塔层的整数精度的运动矢量足够精确来初始化运动矢量搜索到下一个更细的层，因此，除了最后一层外，可以去除所有不必要的插值来计算亚像素对齐误差。最后，为了增加运动搜索范围，减少迭代次数，使用最粗层的水平和垂直的一维投影，通过互相关来估计全局运动。这种优化方法时间比原来缩短4倍。（具体内容参见文献[4]）**
- **帧间融合的优化思路 （工程优化指令集、多线程、OpenCL、OpenGL等），帧间融合可以通过工程优化大幅度降低耗时。**

3. 实际速度和效果测试

测试平台：

- **Inter(R) Core(TM) i-10700 CPU @ 2.9GHz**
- **内存（RAM）：16.0 GB**
- **系统：Windows 10 64位操作系统**
- **代码编辑器：CLion**
- **编译器：VS2019**

下面表格中的DIS的处理图像的大小为 960×540, 原始1080P进行2倍下采样

- **OpenCV的版本是4.5.5，调用DIS光流估计的耗时在5~6ms范围；**
- **A代表的优化策略是将源码中`prepareBuffers`函数在`cal`函数计算光流的时候每一帧都会调用一次，其实在视频降噪过程中是没有必要的，由于视频帧的大小没有变化，因此只初始化一次即可（金字塔的每层大小，梯度，x和y方向光流图等等初始化一遍后，直接内存复用即可），仅仅这个操作让耗时降低到2.4~3.0ms，相比于原始DIS光流耗时降低2倍；**
- **B代表光流稠密化时，除了最精细层，其余层不考虑块重叠，减少插值运算**
- **C代表较粗层光流转化为下一层光流初始值时，采用最近邻插值算法。**
- **D代表梯度下降算法中动态化学习率，加快收敛速度。原始的学习率始终为1，我们优化为动态学习率，迭代初期学习率大于1，随着迭代次数的增加，学习率逐渐下调。**

DIS优化手段	OpenCV	A	B	C	D
耗时（ms）	5~6	2.4~3.0	*	*	*

4. 参考文献

https://blog.csdn.net/sgfmby1994/article/details/68489944

Real-Time Video Denoising On Mobile Phones