Sift（scale-invariant feature transform）尺度不变特征变换算法程序实现流程梳理。

1.createInitImage

01.输入：原始图像，是否二倍图像标志，sigma

02.输出：处理后的图像

（如果二倍则输出二倍后然后 sigma 模糊后的图像，否则输出 sigma 模糊后的图像）

03.程序流程

检查输入图像是否是单通道图像，非单通道转换为单通道图像

将图像数据存储类型转换为 short 类型{转换图像前后 alph 为 1 和 beta 为 0，alpha 为图像像素点值的缩放量 SIFT_FIXPT_SCALE =1，beta 为偏移量 alph*pixel+beta }

{ Lowe 默认初始图像的尺度空间 sigma=0.5，二倍图像的尺度空间为 sigma=1.0 }

如果二倍图像：首先二倍图像，然后用 sigma 高斯模糊图像

否则：直接用高斯模糊图像

2.buildGaussPyramid

01.输入：基准图像，存放高斯金字塔图像的 Mat 容器 vector，高斯金字塔的 nOctave 数目

02.输出：空。（实际结果存放在高斯金字塔 vector<Mat>）

03.程序流程：

每一个高斯金字塔尺度空间的尺寸=nOctaveLayers+3

高斯金字塔的总数目= nOctaves*(nOctaveLayers + 3)

{高斯函数性质：$L(x,y,\sigma_2)=G(x,y,\sqrt{\sigma_2^{\,2}-\sigma_1^{\,2}})\otimes L(x,y,\sigma_1)$ }

尺度空间 sigma 计算=$\sigma=\sqrt{\sigma_2^{\,2}-\sigma_1^{\,2}}$

计算每个 octave 下每一层高斯模糊后的图像：其中每一层第一幅图像用上一个高斯金字塔

倒数第三幅图像降采样得到。

3.buildDoGPyramid

01.输入：高斯金字塔 Mat 容器，高斯差分金字塔 Mat 容器

02.输出：空（实际存放在高斯差分金字塔 Mat 容器中）

03.程序流程：

首先根据高斯金字塔总层数和每一个 octave 层数，计算出 octave 数目

差分高斯金字塔的总数目=nOctaves*(nOctaveLayers + 2)

在高斯金字塔每一个 octave 下，用上一副图像减去下一副图像，最终得到高斯差分金字塔

4.findScaleExtrema

01.输入：高斯金字塔 Mat 容器，高斯差分金字塔 Mat 容器，关键点容器 vector<keypoint>

02.输出：空（实际存放在关键点容器中）

03.程序流程

根据高斯金字塔尺寸和每一个 octave 层数，计算 octave 数目

{指定梯度方向直方图 bins，程序中指定 bins=360，所以每一个 bin 是 10 度}

{指定判断特征点像素点的阈值，threshold=2*0.04/每一个 octave 下的数*255}

建立直方图数组 hist[36]

将每一个像素点的像素值先和阈值 threshold 比较，再和周围 26 个像素点值进行比较，判断出极值点{实际计算时去除图像前五行，后五行，前五列，后五列}

如果是极值点：记录下行号，列号，层号，

{设置边缘阈值 edgeThreshold =10}

调用 adjustLocalExtrema 函数，极值点精确定位：

如果返回值是 false， 可能是下面的情况：

001 如果循环 5 次都没定准，则放弃该极值点，

002 如果定准后的像素点值小于阈值，则放弃该极值点

003 如果主曲率的比值大于阈值，则放弃该极值点，

则执行 continue

最终返回结果如果为 true 说明极值点符合要求，且关键点的坐标，尺度空间坐标，size，响应，已经存放到关键点中，

接下来，调用 calcOrientationHist 函数，根据{特征点所在那幅图像，关键点位置，操作邻域半径，直方图像素点高斯加权的 sigma，直方图数组，直方图 bins 数目}，计算出特征点的直方图 hist，以及直方图的最大值 omax。

循环直方图的所有组数 bins

【

{如果当前组 bin 的幅值大于 0.8*omax，且数值大于左右两个组 bin，那么它就是一个极值点的角度，所以关键点可能有多个主方向，不同主方向为不同的关键点}

{由于组代表角度是一个范围，如 1 组角度为 0~9 度，因此还需要对离散梯度方向直方图进行插值拟合处理，以得到更精确的方向角度值，拟合公式为

$B = i + \frac{H(i-1) - H(i+1)}{2(H(i-1) + H(i+1) - 2H(i))} i = 0,,,15$，最终角度计算公式为 $\theta = 360 - 10 \times B$ }

首先提取直方图当前组的前一组和后一组

前一组号 int l = j > 0 ? j - 1 : n - 1;

后一组号 int r2 = j < n - 1 ? j + 1 : 0;

```
bin = j + 0.5*(hist[l] - hist[r2]) / (hist[l] - 2 * hist[j] + hist[r2]);
kpt.angle = 360.0 - (float)((360.0 / n)*bin);
```

对存放当前关键点 keypoints.push_back(kpt);

】

{函数执行到这里说明已经提取出关键点了：信息包括主方向，坐标，尺度坐标}

结束

5.adjustLocalExtrema

01.输入：高斯差分金字塔，关键点，octave 号，层号，行号，列号，极值点阈值 contrastThreshold=0.04，主曲率阈值 edgeThreshold=10

02.输出：是否满足极值点条件

03.程序流程：

图像归一化系数 ima_scale=1/255

一阶偏导数分母系数 deriv_scale=img_scale*(1/(2h)) {对于图像 h=1}

二阶偏导数 dxx, dyy 分母系数 second_deriv_scale= img_scale/(h^2)

二阶偏导数 dxy 分母系数 cross_deriv_scale= img_scale/(4h^2)

循环判断 5 次

【

{极值点是一个三维矢量，它包括极值点所在的尺度σ，尺度图像所在坐标，即 X=(x,y,σ)}

{使用泰勒级数展开式作为拟合函数，需要计算一阶，二阶偏导数}

计算 Dx=(img(r,c+1)- img(r,c-1))* deriv_scale

计算 Dy=(img(r+1,c)- img(r-1,c))* deriv_scale

计算 Dσ=(next(r,c)- prev(r,c))* deriv_scale

计算 Dxx=(img (r,c+1) + img(r,c-1) - img (r, c)*2)* second_deriv_scale

计算 Dyy=(img (r+1,c) + img(r-1,c) - img (r, c)*2)* second_deriv_scale

计算 Dσσ=(next(r,c) + prev(r,c) - img (r, c)*2)* second_deriv_scale

计算 Dxy=(img (r+1,c+1) + img(r-1,c-1) - img (r+1, c-1) - img (r-1, c+1))* cross_deriv_scale

计算 Dx σ =(next (r,c+1) + prev(r,c-1) - next (r, c-1) - prev (r, c+1))* cross_deriv_scale

计算 Dy σ =(next (r+1,c) + prev(r-1,c) - prev (r+1, c) - next (r-1, c))* cross_deriv_scale

{通过 solve 函数计算函数 $\frac{\partial^2 f}{\partial X^2}\hat{X} = -\frac{\partial f}{\partial X}$,确定极值点精确位置}

cv::Vec3f X = H.solve(dD, cv::DECOMP_LU);

确定出层偏移量，行偏移量，列偏移量

If 层偏移量,行偏移量,列偏移量 三者都小于 0.5 那么已经够确定极值点的位置。break 循环

If 三个偏移量有一个超过一个特别大的值 INT_MAX / 3 ，那么就不是极值点 return false

如果不满足上面两个 if，那么重新根据偏移量重新计算行坐标，列坐标，层坐标。

判断行坐标，列坐标，层坐标是否超出限制，如果超出，return false

】

if 循环 5 次仍然没有确定出确切位置，那么 return false

{如果程序执行到这里，说明已经判断出极值点的准确位置,接下来需要判断极值点的极值是否大于阈值 contrastThreshold }

找到极值点所在的差分高斯金字塔的那个 octave 的那个层，

{通过计算一阶偏导数，求出极值点的极值 $f(\hat{X}) = f(X_0) + \frac{\partial f}{\partial X}(\hat{X})$ }

计算 Dx=(img(r,c+1)- img(r,c-1))* deriv_scale

计算 Dy=(img(r+1,c)- img(r-1,c))* deriv_scale

计算 Dσ=(next(r,c)- prev(r,c))* deriv_scale

dD={Dx, Dy, Dσ}

t=dD.dot(cv::Matx31f(xc, xr, xi)) 即 dD 点乘 X_0

极值 contr = img (r, c)*img_scale + t*0.5

if contr < (contrastThreshold/ nOctaveLayers) 极值点小于阈值，return false

{接下来判断主曲率是否满足要求，代入公式 $\frac{Tr(H)^2}{Det(H)}=\frac{(r+1)^2}{r}$ ，

其中 H 为 Hessian 矩阵， $H=\begin{bmatrix} Dxx & Dxy \\ Dxy & Dyy \end{bmatrix}$ }

计算 Dxx=(img (r,c+1) + img(r,c-1) - img (r, c)*2)* second_deriv_scale

计算 Dyy=(img (r+1,c) + img(r-1,c) - img (r, c)*2)* second_deriv_scale

计算 Dxy=(img (r+1,c+1) + img(r-1,c-1) - img (r+1, c-1) - img (r-1, c+1))* cross_deriv_scale

计算 tr=Dxx+Dxy

计算 det=Dxx*Dyy-Dxy*Dxy

{如果满足公式 $\frac{Tr(H)^2}{Det(H)}<\frac{(\text{edgeThreshold}+1)^2}{\text{edgeThreshold}}$，说明满足特征点的主曲率}

if det<=0|| tr*tr*edgeThreshold >= (edgeThreshold + 1)*(edgeThreshold + 1)*det return false

{执行到这里，说明满足极值点的极值条件，主曲率条件，位置也是精确的，说明是特征点，记录下特征点的坐标，octave，size，response}

{首先是坐标，由于每个 octave 是由原始降采样来的，所以转换到原图坐标，需要乘以 2 的 octave 次幂}

```
kpt.pt.x = (c + xc)*(1 << octv);
kpt.pt.y = (r + xr)*(1 << octv);
```

其余的

```
kpt.octave = octv + (layer << 8) + (cvRound((xi + 0.5) * 255) << 16);
kpt.size = sigma*powf(2.0, (layer + xi) / nOctaveLayers)*(1 << octv) * 2;
kpt.response = std::abs(contr);
```

6.calcOrientationHist

01.输入：图像，关键点在该尺度下的坐标，邻域半径，尺度坐标 sigma（之前乘以 2，又乘以 0.5，所以相当于没变，就是尺度坐标），直方图指针，直方图的 bins=n

02.输出：直方图的峰值

03.程序流程：

{邻域半径 radius=3*1.5*sigma }

所以需要统计 len=radius*radius 个像素点

{需要存放 X, Y, Mag(强度), Ori（方向），Weight(权值)五种数据，但是 X 坐标 Mag 共享一段内存。X(Mag),Y, Ori, W 长度分别是 len,len,len,len,直方图长度是 n+4，为了圆周循环操作直方图，n 个柱体前面空出 2 个，后面空出 2 个，所以直方图的长度为 n+4

所以总共需要一个 len*4+n+4 一段内存}

定义一段内存空间 AutoBuffer<float> buf(len * 4 + n + 4);

将直方图数组前 n 个元素清空

{像素梯度的幅值和幅角计算公式是

$$m(x,y)=\sqrt{(L(x+1,y)-L(x-1,y))^2+(L(x,y+1)-L(x,y-1))^2}$$

$$\theta(x,y)=\arctan(\frac{L(x,y+1)-L(x,y-1)}{L(x+1,y)-L(x-1,y)})$$

其中 X 存放 x 方向梯度，Y 存放 y 方向梯度，Mag 存放幅值，Ori 存放方向角，Weight 存放高斯权重 $e^{\frac{-(i*i+j*j)}{2(\sigma*\sigma)}}$ }

通过双层 for 循环，对邻域内所有像素点进行遍历求取 x 和 y 方向梯度，如果遇到遍历超

出图像索引的位置，continue 循环，每遍历一个像素点 k++，

{循环结束计算出 X, Y, W(只是求出指数部分), k}

再通过 opencv 带的函数

```
cv::hal::exp(W, W, len);
cv::hal::fastAtan2(Y, X, Ori, len, true);
cv::hal::magnitude(X, Y, Mag, len);
```

求出 Ori, Mag, W

进行 k 次循环，对每一个计算出来的方向角，将每一个方向角划分到 n 个 bins 中一个 bin，

并按照高斯权值*赋值叠加到 hist 数组对应的元素上

bin = cvRound((n / 360.0)*Ori[k]);temphist[bin] += W[k] * Mag[k];

对 temphist 前面加的两个位置[-2],[-1]，赋值为[n-2],[n-1]

对 temphist 后面加的两个位置[n],[n+1]，赋值为[0],[1]

{为了防止某个梯度方向角度因受到噪声的干扰而突变，需要对直方图进行平滑处理，使用

公式 $H(i)=\frac{h(i-2)+h(i+2)}{16}+\frac{4\times(h(i-1)+h(i+1))}{16}+\frac{6\times h(i)}{16} i=0,,,15$ }

带入上面的公式，计算出平滑后的直方图数组 hist

然后求出还是 hist 中最大值 maxval

return maxval;

7.calcDescriptors

01.输入：高斯金字塔 Mat 容器，关键点容器 vector<KeyPoints>，描述子存放图像，每

一个 octave 下的层数，第一个 octave 号

02.输出：空

03.程序流程：

从关键点 KeyPoints 中分解出 octave 号，层号，尺度坐标

```
cv::KeyPoint kpt = keypoints[i];
int octave = 0xff & kpt.octave;
```

```
int layer = 0xff00 & kpt.octave;
float scale = powf(2.0, -octave);
```

计算出实际尺度坐标，size = kpt.size*scale

计算出在高斯尺度图像中的实际坐标，Point2f ptf(kpt.pt.x*scale, kpt.pt.y*scale);

提取出高斯尺度图像 img = gpyr[(octave - firstOctave)*(nOctaveLayers + 3) + layer]

计算当前特征点的角度 angle = 360.0 - kpt.angle;

根据{特征点所在的图像，关键点坐标，关键点的角度，所在图像的尺度坐标 sigma，

描述子区域宽度 dxd 中的 d=4，描述子直方图组数=8，描述子指针。

}

调用 calcSIFTDescriptor 函数计算描述子

8.calcSIFTDescriptor

01.输入：特征点所在的图像 img，特征点的坐标 ptf，特征点的主方向 ori，特征点的尺度

坐标 scl，描述子区域宽度 d，描述子直方图组数 n，描述子指针

02.输出：空（实际存放在描述子数组中，128 维）

03.程序流程：

坐标四舍五入 pt(cvRound(ptf.x), cvRound(ptf.y));

方向旋转是正弦和余弦值

```
cos_t = cosf(ori*(float)(CV_PI / 180));
sin_t = sin(ori*(float)(CV_PI / 180));
```

角度转换为直方图组号的系数 bins_per_rad = n / 360.0;

高斯权值指数部分中的系数 exp_scale = -1.0 / (d*d*2.0);

直方图统计区域的宽度，即统计的每个正方形的边长 hist_width = 3*scl;

{为了保证特征点具有旋转不变性，还需要以特征点 Wie 中心，将特征点旋转θ，由于对正

方形进行旋转，为了使旋转后的区域包括整个正方形，应该以正方形中心到它的边的最长距

离为半径，即 $r=\frac{3\sigma(d+1)\sqrt{2}}{2}$ }

radius = cvRound(hist_width*1.4142135623730951*(d + 1)*0.5);

{由于正方形边长为 hist_width，所以进行旋转操作时，需要归一化操作}

归一化系数是：

cos_t /= hist_width;
sin_t /= hist_width;

需要计算的像素点数目是 len = (radius * 2 + 1)*(radius * 2 + 1);

直方图的维度是 d*d*n，为了圆周循环前后各留出一个位置，所以长度 histlen = (d + 2)*(d + 2)*(n + 2);

{需要计算每个像素点的方向角，幅值，高斯权值}

{ X 表示 x 方向梯度，Y 表示 y 方向梯度，Mag 表示梯度幅值，Ori 表示方向角，W 表示高斯权值，RBin 表示三维直方图行坐标，CBin 表示三维直方图纵坐标，hist 表示直方图长度分别是 len，len，len，len，len，len，histlen }

先对三维直方图内存空间清零

双层 for 循环计算每个像素点的 X，Y，RBin，CBin，W

【

{计算每个像素点在三维直方图的横纵坐标}

首先将坐标旋转到主方向

float c_rot = j*cos_t - i*sin_t;
float r_rot = j*sin_t + i*cos_t;

{由于三维直方图下面是 d*d，原点在特征点，即正方形中心，不方便寻址计算，需要将原点平移到正方形左下角，且 cos_t,sin_t 之中均包含/hist_width，所以三维直方图地面 d*d 原点平移只需加 d/2}

{三维直方图每一个区域是一块正方形，表示 hist_width*hist_width 像素区域，同样为

了进行三线性插值，需要坐标平移 0.5f}

平移后的坐标是：

```
rbin = r_rot + d / 2 - 0.5f;
cbin = c_rot + d / 2 - 0.5f;
```

计算出新坐标后，需要判断新坐标是否符合在范围内，且实际双层 for 循环原始坐标是否

在原始图像中

如果不在，进行下一次循环；

如果在，

k++
RBin[k] = rbin; CBin[k] = cbin;
X=dx=img(r, c + 1) - img (r, c - 1)
Y=dy=img (r + 1, c) - img (r - 1, c)

W[k] = (c_rot*c_rot + r_rot*r_rot)*exp_scale;

】

双层 for 循环结束，len=k

计算 Ori，Mag，W

```
cv::hal::fastAtan2(Y, X, Ori, len, true);//gradient angle
cv::hal::magnitude(X, Y, Mag, len);//gradient magnitude
cv::hal::exp(W, W, len);//gaussian weight functionn
```

for 循环所有符合条件的 len 个像素点加权到进行三维直方图中

【

{三维直方图包括行坐标，列坐标，组坐标，

地址累加分别先按组，再按列，再按行}

行列坐标：float rbin = RBin[k], cbin = CBin[k];

组坐标：float obin = (Ori[k] - ori)*bins_per_rad;

高斯权值后的幅值：float mag = Mag[k] * W[k];

{将坐标分为整数部分和小数部分，整数部分对应正方体的点，通过整数部分确定索引值，然后通过小数部分确定实际点，小数部分通过三线性插值，将值分配到正方体的 8 个顶点上}

整数部分：

```
int r0 = cvFloor(rbin);
int c0 = cvFloor(cbin);
int o0 = cvFloor(obin);
```

小数部分：

```
rbin -= r0;
cbin -= c0;
obin -= o0;
```

小数部分三线性插值算法

{公式是：$C_{000}=r\times c\times\theta,\ C_{100}=(1-r)\times c\times\theta,\ C_{010}=r\times(1-c)\times\theta,\ldots$ $C_{111}=(1-r)\times(1-c)\times(1-\theta)$ }

```
float v_r1 = mag*rbin, v_r0 = mag - v_r1;
float v_rc11 = v_r1*cbin, v_rc10 = v_r1 - v_rc11;
float v_rc01 = v_r0*cbin, v_rc00 = v_r0 - v_rc01;
float v_rco111 = v_rc11*obin, v_rco110 = v_rc11 - v_rco111;
float v_rco101 = v_rc10*obin, v_rco100 = v_rc10 - v_rco101;
float v_rco011 = v_rc01*obin, v_rco010 = v_rc01 - v_rco011;
float v_rco001 = v_rc00*obin, v_rco000 = v_rc00 - v_rco001;
```

通过整数部分确定索引坐标：idx = ((r0 + 1)*(d + 2) + c0 + 1)*(n + 2) + o0;

然后由上一步确定的 8 个顶点的分量，将值叠加的三维直方图上

```
hist[idx] += v_rco000;
hist[idx + 1] += v_rco001;
hist[idx + (n + 2)] += v_rco010;
hist[idx + (n + 3)] += v_rco011;
hist[idx + (d + 2)*(n + 2)] += v_rco100;
hist[idx + (d + 2)*(n + 2) + 1] += v_rco101;
hist[idx + (d + 3)*(n + 2)] += v_rco110;
hist[idx + (d + 3)*(n + 2) + 1] += v_rco111;
```

】

计算 128 维描述子的值

```
int idx = ((i + 1)*(d + 2) + (j + 1))*(n + 2);
dst[(i*d + j)*n + k] = hist[idx + k];
```

{对特征矢量进行归一化处理，归一化公式：$q_i = \frac{p_i}{\sqrt{\sum_{j=1}^{128} p_j}}, i = 1,2,...,128$

但由于相机饱和以及三维物体表面的不同数量不同角度的光照变化所引起非线性光照依然存在，它会影响幅值，不会影响幅角，为了消除这部分影响，我们需要引入 t=0.2 的阈值，保留 q 中小于 0.2 元素，大于 0.2 的元素，而把大于 0.2 的元素用 0.2 替代}

```
float nrm2 = 0;
len = d*d*n;
for (k = 0; k < len; k++)
{
    nrm2 += dst[k] * dst[k];
}
float thr = std::sqrt(nrm2)*SIFT_DESCR_MAG_THR;
for (i = 0, nrm2 = 0; i < k; i++)
{
    float val = std::min(dst[i], thr);
    dst[i] = val;
    nrm2 += val*val;
}
nrm2 = SIFT_INT_DESCR_FCTR / std::max(std::sqrt(nrm2), FLT_EPSILON);
for (k = 0; k < len; k++)
{
    dst[k] = cv::saturate_cast<uchar>(dst[k] * nrm2);
}
```