canny算子代码

void CreatGauss(double sigma, double \*\*pdKernel, int \*pnWidowSize);

void GaussianSmooth(SIZE sz, LPBYTE pGray, LPBYTE pResult, double sigma);

void Grad(SIZE sz, LPBYTE pGray, int \*pGradX, int \*pGradY, int \*pMag);

void NonmaxSuppress(int \*pMag, int \*pGradX, int \*pGradY, SIZE sz, LPBYTE pNSRst);

void EstimateThreshold(int \*pMag, SIZE sz, int \*pThrHigh, int \*pThrLow, LPBYTE pGray,   
double dRatHigh, double dRatLow);

void Hysteresis(int \*pMag, SIZE sz, double dRatLow, double dRatHigh, LPBYTE pResult);

void TraceEdge(int y, int x, int nThrLow, LPBYTE pResult, int \*pMag, SIZE sz);

void Canny(LPBYTE pGray, SIZE sz, double sigma, double dRatLow,   
double dRatHigh, LPBYTE pResult);

#include "afx.h"   
#include "math.h"   
#include "canny.h"

// 一维高斯分布函数，用于平滑函数中生成的高斯滤波系数   
void CreatGauss(double sigma, double \*\*pdKernel, int \*pnWidowSize)   
{

LONG i;

//数组中心点   
int nCenter;

//数组中一点到中心点距离   
double dDis;

//中间变量   
double dValue;   
double dSum;   
dSum = 0;

// [-3\*sigma,3\*sigma] 以内数据，会覆盖绝大部分滤波系数   
\*pnWidowSize = 1+ 2\*ceil(3\*sigma);

nCenter = (\*pnWidowSize)/2;

\*pdKernel = new double[\*pnWidowSize];

//生成高斯数据   
for(i=0;i<(\*pnWidowSize);i++)   
{   
dDis = double(i - nCenter);   
dValue = exp(-(1/2)\*dDis\*dDis/(sigma\*sigma))/(sqrt(2\*3.1415926)\*sigma);   
(\*pdKernel)[i] = dValue;   
dSum+=dValue;

}   
//归一化   
for(i=0;i<(\*pnWidowSize);i++)   
{   
(\*pdKernel)[i]/=dSum;   
}

}

//用高斯滤波器平滑原图像   
void GaussianSmooth(SIZE sz, LPBYTE pGray, LPBYTE pResult, double sigma)   
{   
LONG x, y;   
LONG i;

//高斯滤波器长度   
int nWindowSize;

//窗口长度   
int nLen;

//一维高斯滤波器   
double \*pdKernel;

//高斯系数与图像数据的点乘   
double dDotMul;

//滤波系数总和   
double dWeightSum;

double \*pdTemp;   
pdTemp = new double[sz.cx\*sz.cy];

//产生一维高斯数据   
CreatGauss(sigma, &pdKernel, &nWindowSize);

nLen = nWindowSize/2;

//x方向滤波   
for(y=0;y<sz.cy;y++)   
{   
for(x=0;x<sz.cx;x++)   
{   
dDotMul = 0;   
dWeightSum = 0;   
for(i=(-nLen);i<=nLen;i++)   
{   
//判断是否在图像内部   
if((i+x)>=0 && (i+x)<sz.cx)   
{   
dDotMul+=(double)pGray[y\*sz.cx+(i+x)] \* pdKernel[nLen+i];   
dWeightSum += pdKernel[nLen+i];   
}   
}   
pdTemp[y\*sz.cx+x] = dDotMul/dWeightSum;   
}   
}

//y方向滤波   
for(x=0; x<sz.cx;x++)   
{   
for(y=0; y<sz.cy; y++)   
{   
dDotMul = 0;   
dWeightSum = 0;   
for(i=(-nLen);i<=nLen;i++)   
{   
if((i+y)>=0 && (i+y)< sz.cy)   
{   
dDotMul += (double)pdTemp[(y+i)\*sz.cx+x]\*pdKernel[nLen+i];   
dWeightSum += pdKernel[nLen+i];   
}   
}   
pResult[y\*sz.cx+x] = (unsigned char)dDotMul/dWeightSum;   
}   
}

delete []pdKernel;   
pdKernel = NULL;

delete []pdTemp;   
pdTemp = NULL;

}

// 方向导数,求梯度   
void Grad(SIZE sz, LPBYTE pGray,int \*pGradX, int \*pGradY, int \*pMag)   
{   
LONG y,x;

//x方向的方向导数   
for(y=1;y<sz.cy-1;y++)   
{   
for(x=1;x<sz.cx-1;x++)   
{   
pGradX[y\*sz.cx +x] = (int)( pGray[y\*sz.cx+x+1]-pGray[y\*sz.cx+ x-1] );   
}   
}

//y方向方向导数   
for(x=1;x<sz.cx-1;x++)   
{   
for(y=1;y<sz.cy-1;y++)   
{   
pGradY[y\*sz.cx +x] = (int)(pGray[(y+1)\*sz.cx +x] - pGray[(y-1)\*sz.cx +x]);   
}   
}

//求梯度

//中间变量   
double dSqt1;   
double dSqt2;

for(y=0; y<sz.cy; y++)   
{   
for(x=0; x<sz.cx; x++)   
{   
//二阶范数求梯度   
dSqt1 = pGradX[y\*sz.cx + x]\*pGradX[y\*sz.cx + x];   
dSqt2 = pGradY[y\*sz.cx + x]\*pGradY[y\*sz.cx + x];   
pMag[y\*sz.cx+x] = (int)(sqrt(dSqt1+dSqt2)+0.5);   
}   
}   
}

//非最大抑制   
void NonmaxSuppress(int \*pMag, int \*pGradX, int \*pGradY, SIZE sz, LPBYTE pNSRst)   
{   
LONG y,x;   
int nPos;

//梯度分量   
int gx;   
int gy;

//中间变量   
int g1,g2,g3,g4;   
double weight;   
double dTmp,dTmp1,dTmp2;

//设置图像边缘为不可能的分界点   
for(x=0;x<sz.cx;x++)   
{   
pNSRst[x] = 0;   
pNSRst[(sz.cy-1)\*sz.cx+x] = 0;

}   
for(y=0;y<sz.cy;y++)   
{   
pNSRst[y\*sz.cx] = 0;   
pNSRst[y\*sz.cx + sz.cx-1] = 0;   
}

for(y=1;y<sz.cy-1;y++)   
{   
for(x=1;x<sz.cx-1;x++)   
{   
//当前点   
nPos = y\*sz.cx + x;

//如果当前像素梯度幅度为0，则不是边界点   
if(pMag[nPos] == 0)   
{   
pNSRst[nPos] = 0;   
}   
else   
{   
//当前点的梯度幅度   
dTmp = pMag[nPos];

//x,y方向导数   
gx = pGradX[nPos];   
gy = pGradY[nPos];

//如果方向导数y分量比x分量大，说明导数方向趋向于y分量   
if(abs(gy) > abs(gx))   
{   
//计算插值比例   
weight = fabs(gx)/fabs(gy);

g2 = pMag[nPos-sz.cx];   
g4 = pMag[nPos+sz.cx];

//如果x,y两个方向导数的符号相同   
//C 为当前像素，与g1-g4 的位置关系为：   
//g1 g2   
// C   
// g4 g3   
if(gx\*gy>0)   
{   
g1 = pMag[nPos-sz.cx-1];   
g3 = pMag[nPos+sz.cx+1];   
}

//如果x,y两个方向的方向导数方向相反   
//C是当前像素，与g1-g4的关系为：   
// g2 g1   
// C   
// g3 g4   
else   
{   
g1 = pMag[nPos-sz.cx+1];   
g3 = pMag[nPos+sz.cx-1];   
}   
}

//如果方向导数x分量比y分量大，说明导数的方向趋向于x分量   
else   
{   
//插值比例   
weight = fabs(gy)/fabs(gx);

g2 = pMag[nPos+1];   
g4 = pMag[nPos-1];

//如果x,y两个方向的方向导数符号相同   
//当前像素C与 g1-g4的关系为   
// g3   
// g4 C g2   
// g1   
if(gx \* gy > 0)   
{   
g1 = pMag[nPos+sz.cx+1];   
g3 = pMag[nPos-sz.cx-1];   
}

//如果x,y两个方向导数的方向相反   
// C与g1-g4的关系为   
// g1   
// g4 C g2   
// g3   
else   
{   
g1 = pMag[nPos-sz.cx+1];   
g3 = pMag[nPos+sz.cx-1];   
}   
}

//利用 g1-g4 对梯度进行插值   
{   
dTmp1 = weight\*g1 + (1-weight)\*g2;   
dTmp2 = weight\*g3 + (1-weight)\*g4;

//当前像素的梯度是局部的最大值   
//该点可能是边界点   
if(dTmp>=dTmp1 && dTmp>=dTmp2)   
{   
pNSRst[nPos] = 128;   
}   
else   
{   
//不可能是边界点   
pNSRst[nPos] = 0;   
}   
}   
}   
}   
}   
}

// 统计pMag的直方图，判定阈值   
void EstimateThreshold(int \*pMag, SIZE sz, int \*pThrHigh, int \*pThrLow, LPBYTE pGray,   
double dRatHigh, double dRatLow)   
{   
LONG y,x,k;

//该数组的大小和梯度值的范围有关，如果采用本程序的算法   
//那么梯度的范围不会超过pow(2,10)   
int nHist[256];

//可能边界数   
int nEdgeNum;

//最大梯度数   
int nMaxMag;

int nHighCount;

nMaxMag = 0;

//初始化   
for(k=0;k<256;k++)   
{   
nHist[k] = 0;   
}   
//统计直方图,利用直方图计算阈值   
for(y=0;y<sz.cy;y++)   
{   
for(x=0;x<sz.cx;x++)   
{   
if(pGray[y\*sz.cx+x]==128)   
{   
nHist[pMag[y\*sz.cx+x]]++;   
}   
}   
}

nEdgeNum = nHist[0];   
nMaxMag = 0;

//统计经过“非最大值抑制”后有多少像素   
for(k=1;k<256;k++)   
{   
if(nHist[k] != 0)   
{   
nMaxMag = k;   
}

//梯度为0的点是不可能为边界点的   
//经过non-maximum suppression后有多少像素   
nEdgeNum += nHist[k];

}

//梯度比高阈值\*pThrHigh 小的像素点总书目   
nHighCount = (int)(dRatHigh \* nEdgeNum + 0.5);

k=1;   
nEdgeNum = nHist[1];

//计算高阈值   
while((k<(nMaxMag-1)) && (nEdgeNum < nHighCount))   
{   
k++;   
nEdgeNum += nHist[k];   
}

\*pThrHigh = k;

//低阈值   
\*pThrLow = (int)((\*pThrHigh) \* dRatLow + 0.5);

}

//利用函数寻找边界起点   
void Hysteresis(int \*pMag, SIZE sz, double dRatLow, double dRatHigh, LPBYTE pResult)   
{   
LONG y,x;

int nThrHigh,nThrLow;

int nPos;   
//估计TraceEdge 函数需要的低阈值，以及Hysteresis函数使用的高阈值   
EstimateThreshold(pMag, sz,&nThrHigh,&nThrLow,pResult,dRatHigh,dRatLow);

//寻找大于dThrHigh的点，这些点用来当作边界点，   
//然后用TraceEdge函数跟踪该点对应的边界   
for(y=0;y<sz.cy;y++)   
{   
for(x=0;x<sz.cx;x++)   
{   
nPos = y\*sz.cx + x;

//如果该像素是可能的边界点，并且梯度大于高阈值，   
//该像素作为一个边界的起点   
if((pResult[nPos]==128) && (pMag[nPos] >= nThrHigh))   
{   
//设置该点为边界点   
pResult[nPos] = 255;   
TraceEdge(y,x,nThrLow,pResult,pMag,sz);   
}

}   
}

//其他点已经不可能为边界点   
for(y=0;y<sz.cy;y++)   
{   
for(x=0;x<sz.cx;x++)   
{   
nPos = y\*sz.cx + x;

if(pResult[nPos] != 255)   
{   
pResult[nPos] = 0;   
}   
}   
}   
}

//根据Hysteresis 执行的结果，从一个像素点开始搜索，搜索以该像素点为边界起点的一条边界的   
//一条边界的所有边界点，函数采用了递归算法   
// 从（x,y)坐标出发，进行边界点的跟踪，跟踪只考虑pResult中没有处理并且可能是边界   
// 点的像素（=128），像素值为0表明该点不可能是边界点，像素值为255表明该点已经是边界点

void TraceEdge(int y, int x, int nThrLow, LPBYTE pResult, int \*pMag, SIZE sz)   
{   
//对8邻域像素进行查询   
int xNum[8] = {1,1,0,-1,-1,-1,0,1};   
int yNum[8] = {0,1,1,1,0,-1,-1,-1};

LONG yy,xx,k;

for(k=0;k<8;k++)   
{   
yy = y+yNum[k];   
xx = x+xNum[k];

if(pResult[yy\*sz.cx+xx]==128 && pMag[yy\*sz.cx+xx]>=nThrLow )   
{   
//该点设为边界点   
pResult[yy\*sz.cx+xx] = 255;

//以该点为中心再进行跟踪   
TraceEdge(yy,xx,nThrLow,pResult,pMag,sz);   
}   
}   
}

// Canny算子   
void Canny(LPBYTE pGray, SIZE sz, double sigma, double dRatLow,   
double dRatHigh, LPBYTE pResult)   
{   
//经过高斯滤波后的图像   
LPBYTE pGaussSmooth;

pGaussSmooth = new unsigned char[sz.cx\*sz.cy];

//x方向导数的指针   
int \*pGradX;   
pGradX = new int[sz.cx\*sz.cy];

//y方向   
int \*pGradY;   
pGradY = new int[sz.cx\*sz.cy];

//梯度的幅度   
int \*pGradMag;   
pGradMag = new int[sz.cx\*sz.cy];

//对原图高斯滤波   
GaussianSmooth(sz,pGray,pGaussSmooth,sigma);

//计算方向导数和梯度的幅度   
Grad(sz,pGaussSmooth,pGradX,pGradY,pGradMag);

//应用非最大抑制   
NonmaxSuppress(pGradMag,pGradX,pGradY,sz,pResult);

//应用Hysteresis，找到所有边界   
Hysteresis(pGradMag,sz,dRatLow,dRatHigh,pResult);

delete[] pGradX;   
pGradX = NULL;   
delete[] pGradY;   
pGradY = NULL;   
delete[] pGradMag;   
pGradMag = NULL;   
delete[] pGaussSmooth;   
pGaussSmooth = NULL;

}

/\*   
void CChildWnd::OnCanny()   
{   
if (! m\_fOpenFile)   
{   
return;   
}   
m\_fDone = TRUE;   
RGBToGray(szImg, aRGB, aGray, BPP);   
Canny(aGray,szImg,0.1,0.9,0.76,aBinImg);

ShowGrayImage("l",szImg,aBinImg);   
}   
//\*/  
参考资料：贾云德 机器视觉\章毓晋 图象处理和分析\郎锐 数字图象处理学\王娜 李霞 一种新的改进Canny边缘检测算法