# 图像去雾 SA20174043 熊宝霖

一、编程平台和语言

本次编程所使用的平台是Visual Studio 2019，使用的语言为C++。并调用了Opencv库用于对图像进行计算和处理。

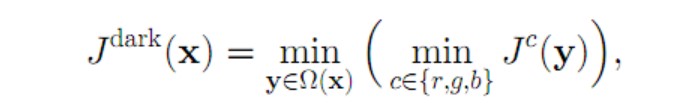
二、算法实现及效果

图像去雾算法主要有以下几个步骤：

1. 计算原始有雾图像的暗通道矩阵
2. 通过计算得到的暗通道，估算大气光值；
3. 通过暗通道和大气光值，估算透射率矩阵；
4. 最后计算复原图像。

下面我将分步介绍我是如何实现以上步骤并展示每个步骤及最终得到的结果。  
 2.1. 计算暗通道

计算暗通道的公式如下：



主要实现代码如下：

void Darkchannels(Mat& input, Mat& output, int patchsize)  
{  
 int rows = input.rows; //行数  
 int cols = input.cols; //列数  
 int paddel = patchsize / 2; //需要补全的边  
 double patch\_min = 255; //指向最小的值  
 double x; //暂存  
 double min\_channels[3]; //三个通道的最小值  
 double max\_channels[3]; //三个通道的最大值  
 Mat temp[3]; //temp矩阵  
 Mat patch; //分布的patch  
 Mat in\_channels[3]; //将temp分为三个通道  
 Rect rect\_patch; //ROI区域  
 Rect rect\_copy(paddel, paddel, cols, rows);  
 split(input, in\_channels);  
 for (int k = 0; k < 3; k++)  
 {  
 temp[k] = Mat::ones(rows + patchsize, cols + patchsize, CV\_8UC1) \* 255;  
 in\_channels[k].copyTo(temp[k](rect\_copy));  
 }   
 rect\_patch.width = patchsize;  
 rect\_patch.height = patchsize;  
 for (int i = 0; i < rows-1; i++)  
 {  
 for (int j = 0; j < cols-1; j++)  
 {  
 rect\_patch.x = j;  
 rect\_patch.y = i;  
 for (int k = 0; k < 3; k++) //分别计算每个通道的最小值  
 {  
 temp[k](rect\_patch).copyTo(patch);  
 minMaxLoc(patch, &min\_channels[k], &max\_channels[k], NULL,NULL);  
 }  
 patch\_min = min(min\_channels[0],min\_channels[1],min\_channels[2]); //返回三个通道中最小值的最小值  
 output.at<uchar>(i, j) = patch\_min; //并把其赋值给output中对应的元素  
 }  
 }  
}

在计算每个通道的最小值的同时也对其进行了最小值滤波。对例子中的图片求暗通道，如下图所示：



图1 暗通道图像

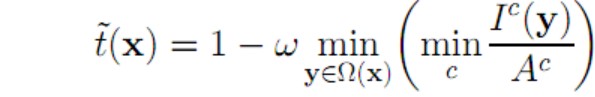
2.2. 估算大气光值

对大气光值的估算，论文中的取法是对于暗通道中前0.1%像素值的像素所对应的原图像像素的最大值。但是关于是对每个通道分别计算最大值，还是三个通道合起来算一个最大值作者说的并不明确。在对两种方法进行对比后，我最终选择后者。 具体的实现算法如下：

for (int i = 0; i < rows; i++)  
 {  
 for (int j = 0; j < cols; j++) //估计自然光  
 {  
 for (int k = 0; k < 3; k++)  
 {  
 if (min\_img\_dark.at<uchar>(i, j) >= thread)  
 {  
 max\_dark = frog\_img.at<Vec3b>(i, j)[k];  
 if (max\_dark > atmosp)atmosp = max\_dark;  
 }  
 }  
 }  
 }

2.3. 估算透射率矩阵

透射率矩阵根据论文中的公式进行计算，公式如下图所示：



实现算法如下：

void Transmission(Mat& dark\_channels, Mat& transmission, int ax, double w)  
{  
 double n;  
 double m;  
 for (int i = 0; i < dark\_channels.rows; i++)  
 {  
 for (int j = 0; j < dark\_channels.cols; j++)  
 {  
 m = dark\_channels.at<uchar>(i, j);  
 n = (ax - w \* m);  
 ;  
 transmission.at<uchar>(i, j) = n;  
 }  
 }  
}

在计算投射率矩阵时，将除以大气光值的部分放到了复原图像的部分，以保持图像矩阵数据类型的一致。计算得到的透射率矩阵如下:

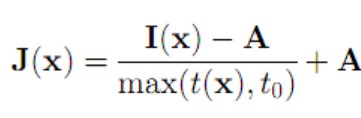
  

图2 透射率矩阵图

2.4. 图像复原

在得到了大气光估计值和透射率矩阵后，就可以计算复原图像，公式如下：



图像复原的主要算法如下图所示：

double tx[3];  
 double src;  
 double irc;  
 for (int k = 0; k < 3; k++)  
 {  
 DeFrog\_img\_channels[k].create(rows, cols, CV\_8UC1);  
 }  
 for (int i = 0; i < rows; i++)  
 {  
 for (int j = 0; j < cols; j++)  
 {  
 for (int k = 0; k < 3; k++)  
 {  
 tx[k] = transmission.at<uchar>(i, j);  
 tx[k] = tx[k] / atmosp;   
 if (tx[k] < 0.1)tx[k] = 0.1; //对投射率进行一定的处理  
 src = frog\_img\_channels[k].at<uchar>(i, j);  
 irc = (src - atmosp) / tx[k] + atmosp;  
 irc = int(irc);  
 if (irc > 255)irc = 255;  
 if (irc < 0)irc = 0;  
 DeFrog\_img\_channels[k].at<uchar>(i, j) = irc;  
 }  
 }  
 }  
 merge(DeFrog\_img\_channels,3,DeFrog\_img);

在图像复原中，我选用了论文中作者给出的0.1阈值对透射率矩阵进行处理。为防止出溢出问题，对大于255的值和小于0的值进行处理。最终的结果如图所示：

图3 去雾图像

三、讨论与总结

1. 投射矩阵参数对图像质量的影响

为了研究透射矩阵中的w参数对图像恢复质量的影响，我分别选择w=0,0.5,0.8.0.95,1对原图像进行去雾处理，控制其他参量不变，得到如下所示的结果：

图4 从左往右依次为w=0,0.5,0.8,0.95和1时的去雾图像

可以看到w的值越大，图像中原来比较趋于白色的区域在恢复图像中会变得越暗，相对来说去雾的效果比较好，但是有比较严重的色彩的畸变。w的值越小，去雾的效果越不明显，当w=0时，相当于没有进行去雾操作，但是能高度保留原图像的色彩。因此，比较之后选择w=0.8可以达到比较好的视觉效果。

1. 总结

通过此次编程实验，我对图像去雾算法有了更深的理解，对使用C++进行图像处理操作更加熟练。目前虽然很多图像处理的工作可以通过人工智能的方式实现，但是传统的图像处理的理论与方法依然具有非常重要的地位，我也会在以后的学习中继续认真探索，将理论与实践相结合，达到理想的学习效果。