OpenCV 实现本地及实时视频磨皮与滤镜

18120189，林艺珺

摘　要：使用 OpenCV 实现自用的本地及实时视频磨皮与滤镜的应用，可实时预览，可自行调参。通过对于常用图像处理软件的磨皮美颜算法的研究、参考相关资料进行算法实现；通过参考常见滤镜算法及常用滤镜效果在图像处理软件的复刻，自制滤镜色彩查找表进行滤镜实现。最终结果视频直接保存本地。

关键词：视频处理，人像美颜，实时视频，双边滤波，滤镜计算

1. 项目背景和意义

前段时间与好友一起录制云端的自媒体节目，介于对画质、收音等要求，使用观感更好的微单进行录制。我个人的相机是 Fujifilm XT3，虽然机器本身带有胶片模拟独占功能，其他品牌能够调节的相关参数也都可以完成，但是由于不是 vlog 专用机，所以无法在拍摄时实时进行磨皮操作，这样会导致后期操作增多。另外，虽然有许多预置的插件，但多数都不符合个人的需求，市面上许多美颜产品都极度失真，失去了视频记录的本质，可调参数也极其有限。关于滤镜方面，个人十分喜欢的手机摄影 App 是通过视觉算法达到已经停产的相机效果的，但并没有一致的视频滤镜，自己通过软件复刻后，希望也可以应用到这个自制程序中。

本着尝试完成一个自己专属的、减轻工作量的可用程序的想法，进行了大作业的计划与实施。后期意识到这或许也就是手机软件的某些原理，自己也可以由此开发或优化一款类似的软件。

2. 项目最初构想

希望完成一个主要基于 Python 以及 OpenCV 库的单机程序，能够完成以下功能：

（1）对于本地视频进行磨皮美白的后期处理，程度可视化可调，可实时预览；

（2）对于笔记本摄像头实时采集视频进行磨皮美白，并提供实时预览，程度可视化可调；

（3）查询相关可参考滤镜公式或色彩查找表，应用于采集画面，可视化选择，实时预览；

（4）保存处理后视频到本地；

（*）相机画面采集问题通过采集卡硬件解决（暂无硬件，理论可实现）。

3. 项目相关原理

3.1 双边滤波

双边滤波的定义如下：

$$I^{filtered}(x) = \frac{1}{W_p}\sum_{x_i\in\Omega} I(x_i) f_r(|I(x_i)-I(x)|) g_s(|x_i-x|)$$

其中$W_p = \sum_{x_i\in\Omega} f_r(|I(x_i)-I(x)|) g_s(|x_i-x|)$，即$I^{filtered}(x) = \frac{\sum_{x_i\in\Omega} I(x_i) f_r(|I(x_i)-I(x)|) g_s(|x_i-x|)}{\sum_{x_i\in\Omega} f_r(|I(x_i)-I(x)|) g_s(|x_i-x|)}$。

$I^{filtered}$是滤波后图像；I是原始图像；x是当前待处理像素坐标；f_r是像素范围核；g_s是空间核；权重W_p依靠空间接近度（g_s空间核）和强度差（f_r像素范围核）决定，假设f_r与g_s都是高斯核，则指定的像素(k,l)为(i,j)降噪是由以下公式决定的：

$$w(i,j,k,l) = \exp\left(-\frac{(i-k)^2+(j-l)^2}{2\sigma_d^2} - \frac{|I(i,j)-I(k,l)|^2}{2\sigma_r^2}\right)$$

其中σ_d与σ_r是平滑参数，随着空间参数σ_d的增加，更多的特征变得平滑；随着范围参数σ_r的增加，双边滤波器会逐渐更接近高斯卷积。进行标准化后，最终得到(i,j)的滤波结果：

$$I_D(i,j) = \frac{\sum_{k,l} I(k,l)w(i,j,k,l)}{\sum_{k,l} w(i,j,k,l)}$$

3.2 HSV 色域选定遮罩

HSV 表示颜色相对于 RGB、CMYK 等模型更加自然，特别是对于色彩范围选取的时候，HSV 更能明确表达目标颜色范围。

OpenCV 中 HSV 的色值范围各是色相 Hue（0-180）、饱和度 Saturation（0-255）、明度 Value（0-255），使用色彩软件进行目标颜色提取，随后经过换算得到 OpenCV 的数据。使用 inRange 来处理目标的实际定位，输出二值图的遮罩。

3.3 滤镜效果处理

滤镜效果一般分为两种实现：一种是提取原图片的 RGB 值，经过某些对于各值的再次计算与变化实现；另一种是通过 Look Up Table 进行色彩的对照变换，通过提取原图片的 RGB 值，在表中寻找到对照色值进行变换。它是一张 512*512 大小的图像，在某些软件中可以直接导入进行滤镜处理。

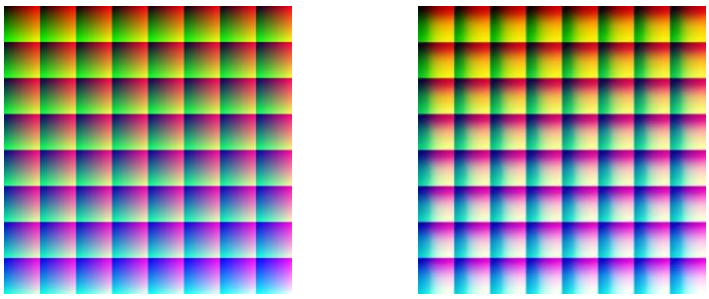

图 1. Look Up Table（左：原始图；右：滤镜效果）

4. 项目实际实现

4.1 磨皮美颜公式及代码

参考网络上对于某商业软件磨皮效果的算法探索，根据其公式，进行些许自适应调整后，转换为程序语言进行实现。原始公式如下：

$$Dst = Src * (1 - P) + (Src + 2 * GuassBlur(EPFFilter(Src) - Src + 128) - 256) * P$$

其中 EPF Filter 指边缘保留滤波，此处使用双边滤波实现。Src 为原始图像，P 为叠加权重比。

在实际实现时，考虑到实际效果，并没有按照比例进行叠加，而是直接使用遮罩和反遮罩进行了矩阵的加运算，具体代码实现如下：

```
1  value1 = cv.getTrackbarPos(trackbar1, windowName)
2  value2 = 1
3  dx = int(value1)
4  fc = value1 * 2.5
5  dst = frame + 2 * cv.GaussianBlur((cv.bilateralFilter(frame, dx, fc, fc) - frame + 128), (2 * value2 -
   1, 2 * value2 - 1), 0, 0) - 255
6  face = cv.add(dst, np.zeros(np.shape(frame), dtype=np.uint8), mask=mask)
7  frame = cv.add(frame, np.zeros(np.shape(frame), dtype=np.uint8), mask=cv.bitwise_not(mask))
8  frame = cv.add(face, frame)
```

getTrackbarPos 是获取窗口中拖动条的值，通过对于窗口中值条的拖动加强或减弱美颜等级，此处根据先前测试限制值为 0-10 之间。value2 是高斯模糊的 dsize 的参考值，最终的 dsize 大小由(2 * value – 1)决定。根据 OpenCV 文档中 bilateralFilter 的描述，设置参数，frame 为原始视频截帧，dx 为像素的邻域直径，fc 为两个 sigma 的参数，这里为了方便根据倍数设置成相同的。由于 OpenCV-Python 没有 copyTo 方法，因此使用两个相反的遮罩进行 add 方法操作，得到磨皮美颜后结果。

4.2 具体 HSV 色域选择及图像处理

图 2. 色彩软件拾色图

所需磨皮美颜部分根据 3.2 中描述的过程进行选定。首先对目标画面截图进行色彩提取，并根据色彩软件提取的结果，进行换算后得到如下区间值：

```
faceLower = (0, 50, 145)
faceUpper = (180, 115, 230)
```

根据该区间值，使用 inRange 方法得到二值图的遮罩，供 4.1 中使用。结果经过形态学操作，消除遮罩中的噪声。图像的膨胀操作将图像与任意形状内核进行卷积，将内核覆盖区域的最大像素值提取，代替锚点（内核中心点）的像素值；腐蚀操作基本同理，只是使用最小像素值填充。

```
mask = cv.inRange(hsv, faceLower, faceUpper)
mask = cv.erode(mask, None, iterations=2)
mask = cv.dilate(mask, None, iterations=2)
```

另外使用 bitwise_not 方法对遮罩进行取反，方便进行矩阵加的操作。

4.3 滤镜效果的公式实现与 Look Up Table 实现

同磨皮美颜效果一样，滤镜效果的切换使用窗口的值条拖动来实现。在程序中预置了 20 个滤镜，其中 0 号为原图滤镜，1、2、3 号使用公式实现，其余使用滤镜色彩查找表实现。

以 3 号流年效果为例，公式实现如下：（其中 R、G、B 均小于等于 255，大于等于 0）

$$\begin{cases} R = R_{ori} \\ G = G_{ori} \\ B = 12\sqrt{B_{ori}} \end{cases}$$

获取截帧的长宽后，遍历每一个像素点，根据公式进行计算与重新赋值，得到滤镜后图像。

Look Up Table 的实现依据其原理，根据当前像素点当前的 RGB 色值，查找表得到相应的变化色彩，并对像素点进行重新赋值，算法封装为一个子函数，实现如下：

```
def getBGR(table, b, g, r):
    x = int(g/4 + int(b/32) * 63)
    y = int(r/4 + int((b%32) / 4) * 63)
    return table[x][y]
```

其中 b、g、r 为从主函数遍历操作中获取的当前像素的 RGB 色值，table 为主函数中使用 imread 方法读取的滤镜色彩查找表，如图 1 所示。

5. 实验结果及分析

本次实验进行了本地视频以及笔记本电脑摄像头实时捕获的视频处理测试。

5.1 本地视频处理

本地视频的处理速度相较于后续进行的实时捕获视频速度相对较快，且因为后续存储使用的是原视频获取的帧率，因此输出视频并不会出现掉帧的情况。猜测是电脑配置的原因，完成一次大约 1 分钟的视频处理可能要半小时左右，具体时间并未测试，基本每次都是手动通过程序设置的 q 键关停，由于存储的设置不会丢失先前已经处理过的视频帧文件。

图 3. 程序窗口美颜 6 级无滤镜实时预览截图

另外在测试中发现滤镜色彩查找表的计算速度比公式慢许多，具体速度每次运行都有差异，因此在测试中不断压缩画幅比例以正常运行。

图 4. 美颜 6 级+某滤镜色彩查找表效果图

图 5. 美颜 6 级+流年效果滤镜公式效果图

5.2 实时视频捕获与处理

由于电脑配置问题，实时捕获的视频帧率无法保证，也无法通过程序获取或修复还原成为正常帧率，因此输出视频效果类似于延时摄影。在加载已经很慢的情况下，对于滤镜的测试只采用了滤镜公式的方式，具体效果如下图。

虽然为了程序运行压缩了不少画面，但是还是能直接看出对于皮肤产生的平滑效果的。经过测试，本程序中 6 级美颜较为自然。实时捕获的 HSV 参数并未修改，直接沿用了 5.1 本地视频的拾色结果，说明本实验先前取色有一定包容性。但是当环境光变化极大时，将无法正确识别肤色，如色彩光、冷白光等，在此种情况下需要根据捕获画面重新拾取皮肤颜色（本地视频光照变化同样适用）。

图 6. 美颜 6 级效果图

图 7. 无美颜效果图

图 8. 美颜 10 级效果图

图 9. 美颜 6 级+流年效果图

图 10. 无美颜+流年效果图

图 11. 美颜 10 级+流年效果图

6. 大作业及课程感想

大作业的构思其实想了挺久的，但是一直没有决定。中途有朋友向我介绍了一个新媒体艺术项目，大致内容是将人像图片灰度化处理后，对一个区域内的像素点进行取最值操作（最大或最小）后，在该区域范围内可视化为一个基本图形，其大小所占区域范围比例为取最值操作后获得的色值在 0-255 的比例位置，最终输出的图像是依据基本图像大小而显示出的有轮廓感的新媒体作品。但是最终想了想，还是决定实现一个有用户群的常见应用的自用版本，并且最初的想法是确确实实要用于自己的视频制作的，但未曾想到电脑配置不足以顺利完成这样的操作，也让我惊叹于现在市面上的商业软件在速度、内存优化上其实有进行很多努力。

原本实现的程序还预想了一个贴图功能，在理论上，通过肤色检测的范围，计算中心后，根据相应的比例，在面部的适当位置增加贴图，但是由于电脑及程序告警许多次，再往下压缩画面大小就无法输出效果，最终不得不放弃。

在制作的过程中，搜寻到了一些同样使用 OpenCV 进行磨皮美颜操作的项目，其中大多数都使用 OpenCV 自带的人脸检测来获取所需磨皮的区域，但是实现的效果并不能使我接受，其中产生了过多的误平滑操作。因此借鉴了实验 9 中 BallTracking 的 HSV 色域范围的方式，进行了操作范围选定。

很庆幸自己选课的时候下了狠心选择了《计算机视觉》，对于没有学习过《数字图像处理》的大二学生来说，有一些理论在第一次接触的时候确实会觉得十分晦涩，很不好意思的是，现在还有些理论类的知识没有完全转化为自己的知识。这次疫情改变了课程的考评方式，但我十分感谢每周一次的实验，虽然作为一名 Python 与 OpenCV 的双料初学者，为此付出了许多的时间，许多个夜晚与各类文档和 Pycharm 共度，但是不得不说，计算机作为一门极需实践的学科，学习到的理论知识要是无法实际应用留下印象的话，是很难有长久牢固的知识把握的。

计算机视觉是现在计算机行业的一大发展方向，近日云毕业轰轰烈烈，腾讯优图顺势推出了一款基于小程序的季节应用，根据用户正脸照片生成正脸或侧脸的毕业照。虽然我个人感觉这款应用的效果并不好，失真程度较高，但是风靡朋友圈的尝试与转发，也在证明计算机视觉正在逐步走近平常生活，包括有许多企业单位都在招聘 CV 算法岗的工程师。作为一个自认对新媒体技术、艺术比较感兴趣的学生爱好者，我想这个方向还有许多值得我去探索的，如果自己能够吸收这些理论知识并寻找到自己的爱好点，我也愿意将此作为我的学术方向，无论我会尝试多久学术的路。

参考文献

[1] Ball Tracking with OpenCV, https://www.pyimagesearch.com/2015/09/14/ball-tracking-with-opencv/

[2] 简单探讨可牛中具有肤质保留功能磨皮算法及实现细节, http://www.cnblogs.com/Imageshop/p/4709710.html

[3] Bilateral Filters（双边滤波算法）原理及实现（一）, https://blog.csdn.net/u013066730/article/details/87859184

[4] OpenCV Document for 4.3.0, https://docs.opencv.org/4.3.0/index.html

[5] 图像特效处理之素描、怀旧、光照、流年以及滤镜特效, https://blog.csdn.net/Eastmount/article/details/99566969

[6] OpenCV 学习十一：dilate、erode 膨胀及腐蚀, https://blog.csdn.net/kakiebu/article/details/79320538