美颜相机实验报告

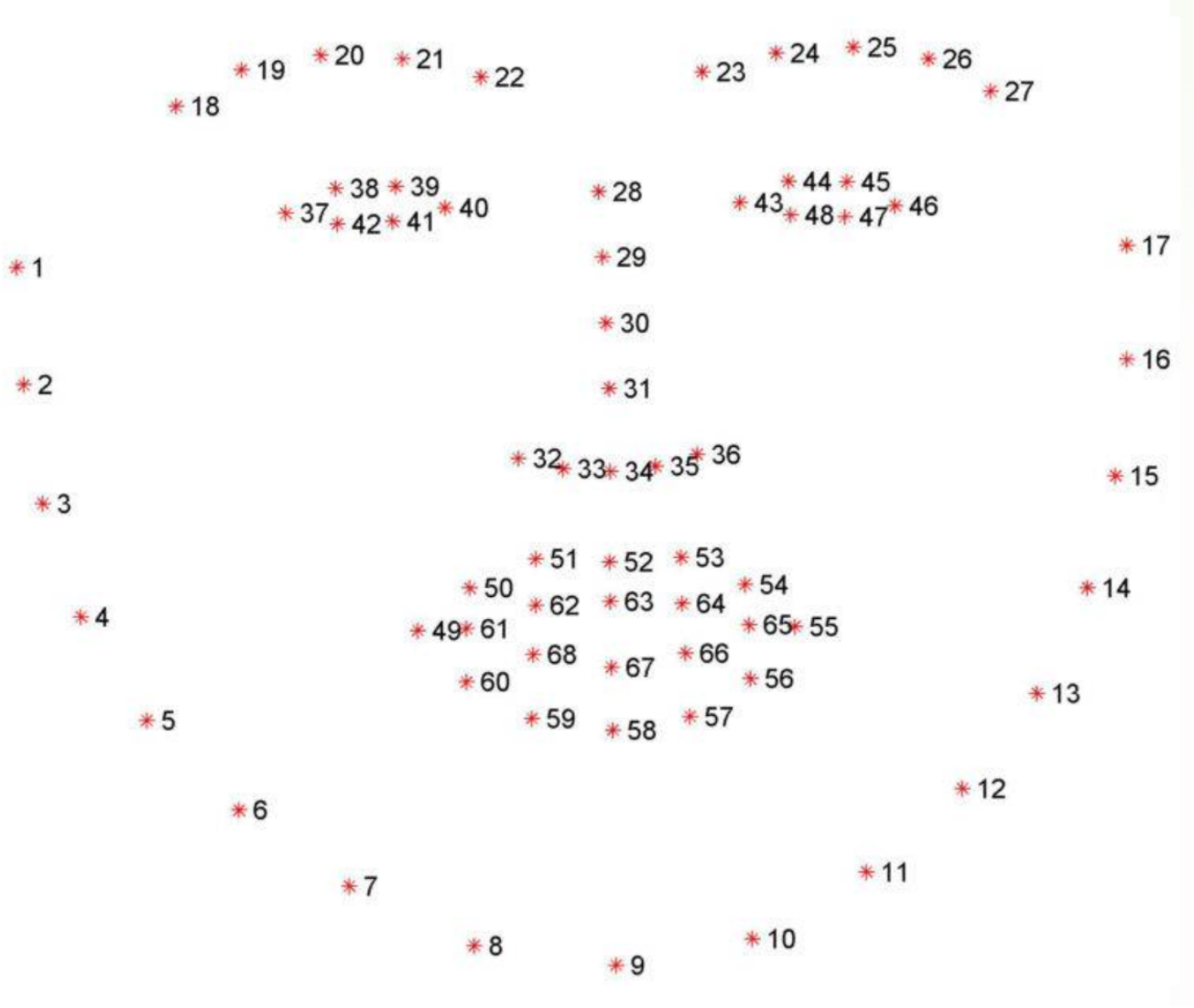
热伊莱·图尔贡 2018K8009929030

实验题目：美颜相机

算法介绍

- dlib人脸算法库

  Dlib库是用于人脸检测的开源工具包。 提供的dlib人脸关键点示例代码可用于确定人脸68个关键点，分布如下图：



- 图像局部变形算法

> http://www.gson.org/thesis/warping-thesis.pdf

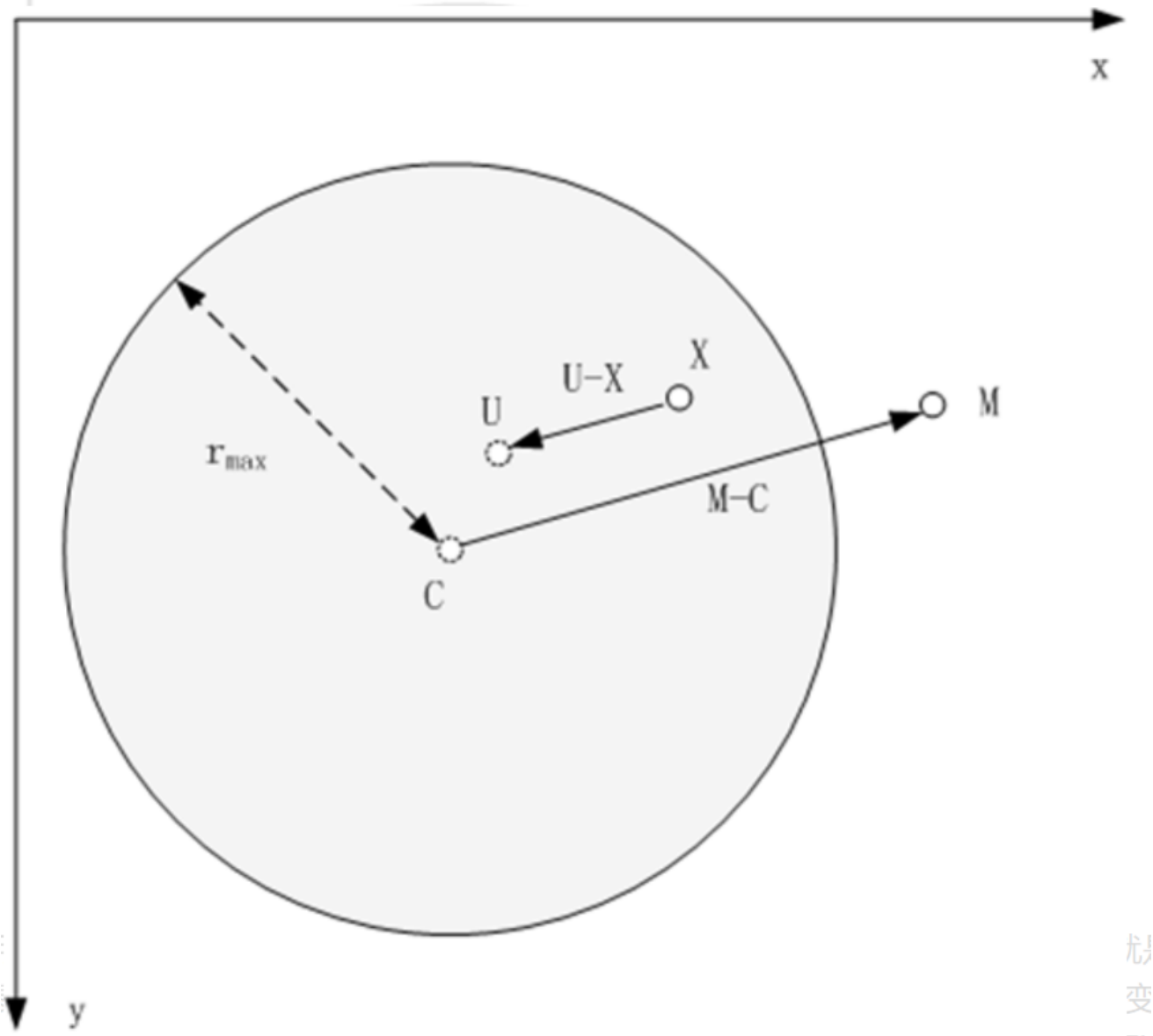
这篇论文详细描述了图像局部变形算法原理，并提供了伪码实现。 图像局部变形算法包括局部缩放（Local Scaling）算法、局部平移（Local Transition）算法和局部旋转（Local Rotation）算法。 其中局部平移算法（Local translation warps）可用于实现瘦脸效果，局部缩放算法（Local scaling warps）可实现眼睛放大效果。

算法各步骤的设计动机

- 美颜算法思路：

1. 通过Dlib进行图片人脸检测和特征点提取
2. 通过图像变形算法对特征点向量进行变形： 由变形后坐标，根据逆变换公式反算变形前坐标，然后插值得到该坐标rgb像素值，将该rgb值作为变形后坐标对应的像素值。

- Interactive Image Warping



阴影圆环代表一个半径为 rmax 的圆形选区。因为是交互式图像局部变形，因此各个点可以看作鼠标的移动。其中，C点是鼠标点下时的点，也就是圆形选区的圆心。鼠标从C拖到M，致使图像中的点U变换到点X。 对于本次实验关键在于找出逆变换，即在给出点X时，可以求出它变换前的坐标U（精确的浮点坐标），然后用变化前图像在U点附近的像素进行插值，求出U的像素值。 对圆形选区内的每一个像素进行求值，即可得出变换后的图像。

- Local translation warps 逆变换公式：

$$\vec{u} = \vec{x} - \left(\frac{{r_{\max}}^2 - |\vec{x} - \vec{c}|^2}{({r_{\max}}^2 - |\vec{x} - \vec{c}|^2) + |\vec{m} - \vec{c}|^2} \right)^2 (\vec{m} - \vec{c})$$

其中x是变换后的位置，u是原坐标位置。整个计算在以c为圆心，r为半径的圆内进行，c和m决定变形方向。

```
def localTranslationWarp(Matimg, startX, startY, endX, endY, radius):
  db_radius = float(radius * radius)
  copyImg = np.zeros(Matimg.shape, np.uint8)
  copyImg = Matimg.copy()
  # 计算|m-c|^2
  db_mc = (endX - startX) * (endX - startX) + (endY - startY) * (endY -
startY)
  H, W, C = Matimg.shape
  for i in range(W):
      for j in range(H):
          # 该点是否在形变圆的范围之内
          # 在 (startX,startY)的矩阵框中?
          if math.fabs(i - startX) > radius and math.fabs(j - startY) >
radius:
              continue
          distance = (i - startX) * (i - startX) + (j - startY) * (j - startY)
          if distance < db_radius:
              # 计算出 (i,j) 坐标的原坐标
              # 计算公式中右边平方号里的部分
              ratio = (db_radius - distance) / (db_radius - distance + db_mc)
              ratio = ratio * ratio
              # 映射原位置
              UX = i - ratio * (endX - startX)
              UY = j - ratio * (endY - startY)
              # 根据双线性插值法得到UX, UY的值
              value = BilinearInsert(Matimg, UX, UY)
              # 改变当前 i , j的值
              copyImg[j, i] = value
  return copyImg
```

- Local scaling warps 逆变换公式：

$$f_s(r) = \left(1 - \left(\frac{r}{r_{\max}} - 1\right)^2 a\right) r$$

实现代码类似

- 瘦脸调整
  - 优化前：对左右脸部下颌部分进行变形

```
for landmarks_node in landmarks:
      # 左脸下颌开始
      left_landmark = landmarks_node[3]
      # 左脸下颌结束
      left_landmark_down = landmarks_node[6]
      # 右脸下颌开始
      right_landmark = landmarks_node[14]
      # 右脸下颌结束
      right_landmark_down = landmarks_node[11]
```

```
        # 鼻子点
        endPt = landmarks_node[30]

        # 计算左脸下颌两端的距离作为瘦脸距离
        r_left = math.sqrt(
            (left_landmark[0, 0] - left_landmark_down[0, 0]) * (left_landmark[0,
0] - left_landmark_down[0, 0]) +
            (left_landmark[0, 1] - left_landmark_down[0, 1]) * (left_landmark[0,
1] - left_landmark_down[0, 1]))

        # 计算右脸下颌两端的距离作为瘦脸距离
        r_right = math.sqrt(
            (right_landmark[0, 0] - right_landmark_down[0, 0]) *
(right_landmark[0, 0] - right_landmark_down[0, 0]) +
            (right_landmark[0, 1] - right_landmark_down[0, 1]) *
(right_landmark[0, 1] - right_landmark_down[0, 1]))

        # 瘦左边脸
        slim_face_image = localTranslationWarp(src, left_landmark[0, 0],
left_landmark[0, 1], endPt[0, 0], endPt[0, 1],
                                               r_left)
        # 瘦右边脸
        slim_face_image = localTranslationWarp(slim_face_image,
right_landmark[0, 0], right_landmark[0, 1], endPt[0, 0],
                                               endPt[0, 1], r_right)
```

结果实例：

- 优化后：从左颧骨到下巴再到右颧骨依次做调整

```
  node = landmarks[0]
    endPt = node[16]
    for i in range(3, 14, 2):
        landmark_start = node[i]
        landmark_end = node[i + 2]
        r = math.sqrt(
            (landmark_start[0, 0] - landmark_end[0, 0]) * (landmark_start[0, 0] -
landmark_end[0, 0]) +
            (landmark_start[0, 1] - landmark_end[0, 1]) * (landmark_start[0, 1] -
landmark_end[0, 1]))
        slim_face_image = localTranslationWarp(slim_face_image, landmark_start[0,
0], landmark_start[0, 1], endPt[0, 0],
                                               endPt[0, 1], r)
```

结果实例：

- 大眼调整

```
    for landmarks_node in landmarks:
        # 左眼皮上开端
        left_landmark = landmarks_node[38]
        # 右眼皮上开端
        right_landmark = landmarks_node[43]
        # 山根
        landmark_down = landmarks_node[27]
        # 鼻尖
        endPt = landmarks_node[30]

        # 计算左眼到山根的距离作为左眼放大范围
        r_left = math.sqrt(
```

```
            (left_landmark[0, 0] - landmark_down[0, 0]) * (left_landmark[0, 0] -
landmark_down[0, 0]) +
            (left_landmark[0, 1] - landmark_down[0, 1]) * (left_landmark[0, 1] -
landmark_down[0, 1]))

        # 计算右眼到山根的距离作为右眼放大范围
        r_right = math.sqrt(
            (right_landmark[0, 0] - landmark_down[0, 0]) * (right_landmark[0, 0]
- landmark_down[0, 0]) +
            (right_landmark[0, 1] - landmark_down[0, 1]) * (right_landmark[0, 1]
- landmark_down[0, 1]))

        # 左眼
        big_eye_image = LocalScalingWarp(src_img, left_landmark[0, 0],
left_landmark[0, 1], endPt[0, 0], endPt[0, 1],
                                         r_left)
        # 右眼
        big_eye_image = LocalScalingWarp(big_eye_image, right_landmark[0, 0],
right_landmark[0, 1], endPt[0, 0],
                                         endPt[0, 1], r_right)
```

- 磨皮效果

```
img = cv2.imread('beautify/face_eye.jpg')
lighten = cv2.bilateralFilter(img, 15, 25, 20)
cv2.imwrite('beautify/6.jpg', lighten)
```

经过多次调试，得出这个数值对大部分人像最自然。

实验结果分析（包括可视化）

原图： 

瘦脸后：


大眼

后： 

磨皮后：

如果需要实时呈现照片可以添加语句

```
cv2.imshow('显示照片名字', 照片变量名)
```

我的程序里没有添加实时显示功能，根据README运行test.py文件即可将美颜后的图片保存于beautify目录下。 也可以分别运行瘦脸，大眼，磨皮程序查看各个阶段的效果。

算法优缺点、适用范围分析

此程序对于一些脸部阴影过大的人像照（例如图九）会无法进行瘦脸操作。 程序效率较低，瘦脸大眼效果自然程度因人而异。 磨皮效果最大程度地追求自然，不油画化。