北京大学

本科实验报告

课程名称：机器感知实验

姓　　名：金镇雄

学　　院：元培

　　系：智能科学系

专　　业：智能科学与技术

年　　级：19

学　　号：1900094619

指导教师：曲天书

职　　称：副教授

2022 年　　6 月　　20 日

基于肤色分割和运动目标分割的人手识别

一、实验目的

- 使用 MATLAB 进行人手检测与跟踪
- 利用多颜色空间模型和 K 均值聚类算法实现肤色分割
- 利用帧差法实现运动目标分割
- 了解形态学处理的原理和作用

二、实验仪器设备

PC 机、MATLAB、电脑摄像头

三、MATLAB 工具箱

- MATLAB Support Package for USB Webcams
- Image Processing Toolbox
- Statistics and Machine Learning Toolbox

四、实验步骤

1. 从电脑摄像头获取当前帧的 RGB 图像
2. 对图像进行左右翻转以及双边滤波。
3. 基于灰度值分布进行图像分割，将前景和背景分离。
4. 肤色分割
   A. 根据每个像素点的 CbCr 分量、HS 分量和 Normalized RG 分量，进行基于多颜色空间模型的肤色分割，初步确定肤色区域。
   B. 结合图像的 Cb,Cr,H 分量和初步的肤色分割结果构建数据集。
   C. 根据从第五步得到的数据集进行 K-means 聚类模型的训练或测试，将图像划分为前景、背景和肤色区域三个聚类。
   D. 对聚类结果进行形态学处理得到最终的肤色分割的二值化图像。
   E. 计算二值化图像中每个连通区域的质心 skinC。
5. 运动目标分割
   A. 用当前帧和上一帧的灰度图进行基于帧差法的运动目标分割。
   B. 计算二值化图像中每个连通区域的质心 moveC。
   C. 用 moveC 计算整个二值化图像的质心 moveC'。
6. 计算 skinC 中每个质心到 moveC'的距离，将其距离最小的质心对应的连通区域作为人手区域。

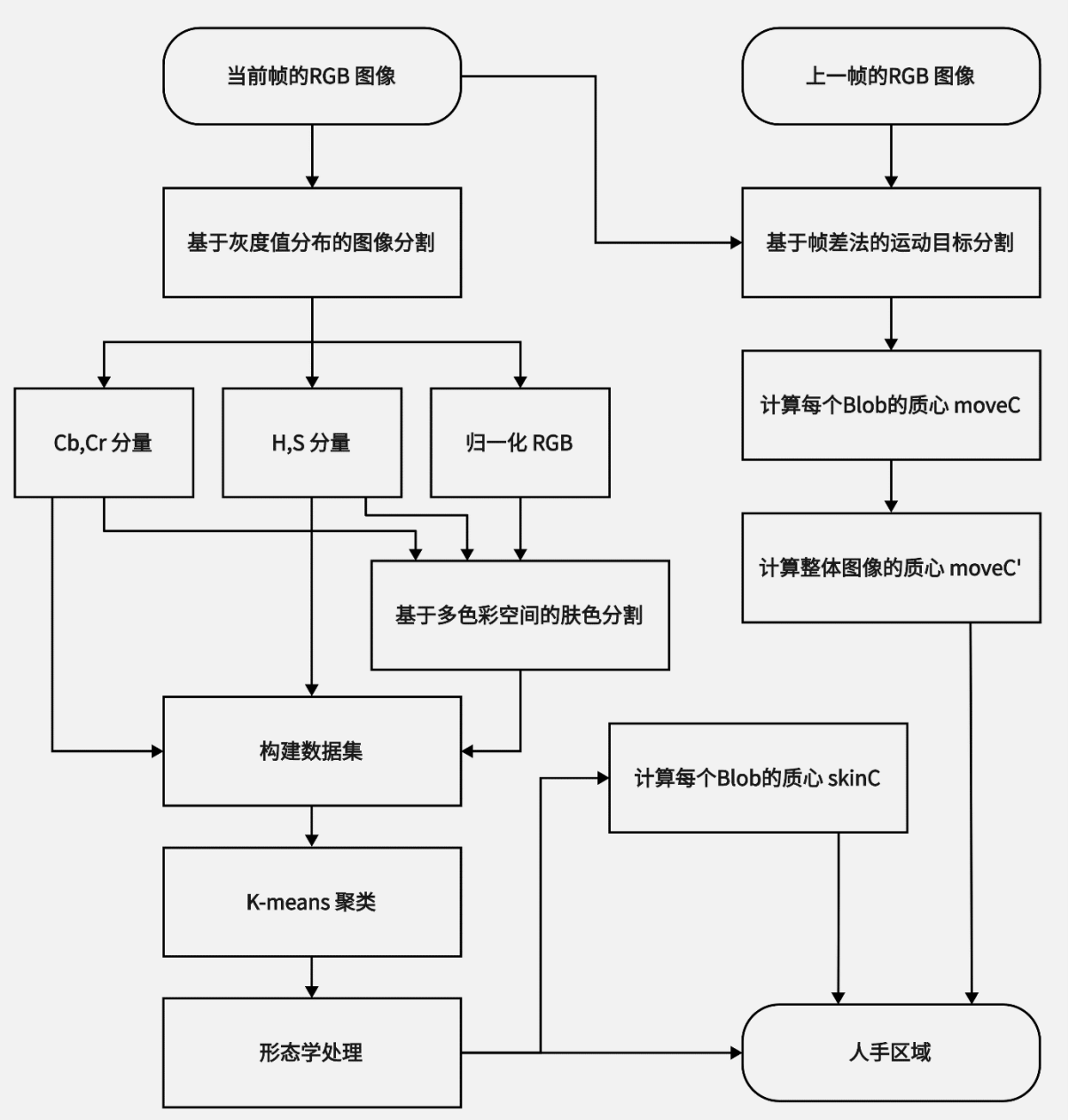


Fig.1. 流程图

五、 实验原理

针对在复杂环境背景下肤色、光照变化等对动态手势分割的干扰，本实验使用了结合基于 K-means 聚类的肤色分割和帧差法的动态手势分割方法。该方法首先把图像的 RGB 颜色空间转为 YCbCr 、HSV 和归一化 RGB 颜色空间，利用肤色检测初步确定肤色区域，然后进行 K-means 聚类再次确定肤色区域，最终结合帧间差分方法对运动的人手进行检测，确定人手区域，以解决光照变化、类肤色背景等因素对手势分割的干扰问题。

1. 基于多颜色空间模型的肤色分割

肤色是人类皮肤重要特征之一，在检测人脸或手等目标时常采用肤色检测的方法，将相关区域从图像中分割出来。肤色检测方法有很多，但无论是基于不同的颜色空间还是不

同的肤色模型，其根本出发点在于肤色分布的聚集性，即肤色的颜色分量一般聚集在某个范围内。因此，我们可以根据大量样本的统计数据建立以确定肤色的分布规律，进而判断像素的色彩是否属于肤色或与肤色相似程度。

在 Human Skin Color Clustering for Face Detection 一文中提出了四种典型而简单的肤色判别算式：基于 RGB 颜色空间的阈值肤色分割、基于 YCrCb 颜色空间 Cr,Cb 范围筛选法、基于 HSV 颜色空间 H 范围筛选法和基于椭圆模型的肤色分割[1]。这些方法依据肤色分布范围进行检测，判断简单、明确、快捷。在比较单一的背景下，它们的分割结果是可靠的。但是如果在背景中有类肤色区域，则这些方法的误检率比较高，很容易将非肤色区域判别为肤色区域。

与基于单一颜色空间的肤色分割相比，基于多颜色空间模型的肤色分割的检测率比较高，误检率非常低。其中，检测率为将肤色像素点正确判别为肤色的概率，误检率为非肤色像素点判别为肤色的概率。在 HGR 数据集上测试各种肤色分割算法发现，CbCr 范围筛选法的平均检测率和误检率分别为 79.18%和 20.8%，HS 范围筛选法的平均检测率和误检率分别为 85.69%和 13.5%，而在 Skin Color Segmentation Using Multi-Color Space Threshold 一文中提出的结合 HSV、YCbCr 和归一化 RGB 颜色空间的 Proposed Method 可以将检测率提高到 95.4%，误检率下降到 1.76%[2]。本实验中以 Proposed Method 进行了对肤色区域的初步判断。

YCbCr 颜色空间是一种常用的肤色检测的色彩模型，其中 Y 代表亮度，Cr 代表光源中的红色分量，Cb 代表光源中的蓝色分量。因为肤色的 YCbCr 颜色空间中 CbCr 分布集中在较小的区域内，并且亮度 Y 不影响对肤色的判断，所以只通过判断当前像素点的 CbCr 是否落在肤色分布的特定区域内，就可以很容易地确认当前像素点是否属于肤色。CbCr 分量的阈值范围由下列两个公式给出：

$$77 \leq Cb \leq 127 \qquad (1)$$

$$133 \leq Cr \leq 173 \qquad (2)$$

HSV 颜色空间也是一种典型的肤色分割的色彩模型。此模型中颜色的参数分别是色调 H、饱和度 S 和明度 V。基于此空间模型的肤色分割方法类似于基于 CbCr 空间模型的分割，对 H 和 S 分量进行一定的阈值处理后可得到很准确的肤色区域。据资料显示，肤色的 H 和 S 分量有以下特性：

$$0.01 \leq H \leq 0.1 \qquad (3)$$

$$0.0754 \leq S \leq 0.6093 \qquad (4)$$

通过对图像的 RGB 颜色空间进行归一化处理，在某些情况下是去除光照和阴影影响的一种简单和有效的方法。因此，基于归一化 RGB 颜色空间的肤色分割也是可以判断是否为肤色的有效方法。假设 RGB 代表原图像某点的像素值，rgb 表示归一化后的值，则

rgb 由以下公式获得：

$$r = \frac{R}{R+G+B} \quad (5)$$

$$g = \frac{G}{R+G+B} \quad (6)$$

$$b = \frac{B}{R+G+B} \quad (7)$$

基于归一化 RGB 模型的肤色分割的判断公式为：

$$\frac{r}{g} > 1.185 \quad (8)$$

$$\frac{R.G}{(R+G+B)^2} > 0.107 \quad (9)$$

$$\frac{R.B}{(R+G+B)^2} > 0.112 \quad (10)$$

基于多颜色空间模型的 Proposed Method 实现步骤如下：

1. 将 RGB 图像转换到 YCbCr 颜色空间，利用公式 1 和公式 2 判断是否为肤色。
2. 将 RGB 图像转换到 HSV 颜色空间，利用公式 3 和公式 4 判断是否为肤色。
3. 将 RGB 图像转换到归一化颜色空间，利用公式 8 进行阈值处理。
4. 结合步骤 1 到 3 的结果

上述步骤的流程图如下：

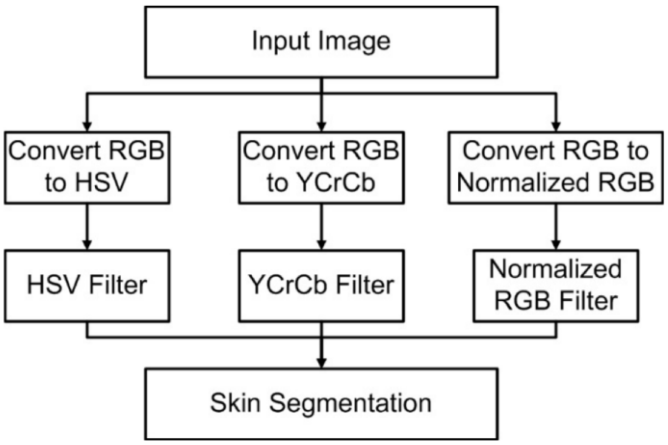


Fig.2. 基于多颜色空间模型的肤色分割的流程图

2. 基于灰度值分布和 K-means 聚类的肤色分割

在 Skin Detection Based on Image Color Segmentation with Histogram and K-Means Clustering 一文中提出了一种基于灰度值分布和 K-means 聚类的肤色分割方法[3]。其中步骤是，在原始图像上进行基于灰度值分布的图像分割，将图像的前景和背景分离，在其前景上做肤色分割初步确定图像中的肤色区域，以图像的 Cr、Cb 和 H 分量以及肤色分割结果构建数据集后进行 K-means 聚类，将图像划分为前景、背景和肤色区域，然后取其中的肤色区域作为最终结果。

基于灰度值分布的图像分割的具体操作如下：将 RGB 图像转换为灰度图，根据公式 11 用 Otsu 阈值 Totsu 和频数最大的灰度值 Tmax 重新计算阈值 η，当 Tmax≤ 220时，根据公式 12 对图像进行二值化处理；否则根据公式 13 做二值化。在公式 12 和 13 中， h(x,y)表示像素点(x,y)的灰度值，1 和 0 分别表示前景和背景。

$$\eta = \begin{cases} round(\frac{T_{Otsu}+T_{max}}{4}), & \text{if } T_{max} \leq 10 \\ round(\frac{T_{Otsu}+T_{max}}{2}, & \text{if } T_{max} > 10 \end{cases} \tag{11}$$

$$g(x,y) = \begin{cases} 0, & \text{if } h(x,y) \leq \eta \\ 1, & \text{if } h(x,y) > \eta \end{cases} \tag{12}$$

$$g(x,y) = \begin{cases} 0, & \text{if } h(x,y) \geq \eta \\ 1, & \text{if } h(x,y) < \eta \end{cases} \tag{13}$$

Fig.3 为一个图像的灰度值分布图，其中，横轴表示灰度值，其范围为 0~255；纵轴表示灰度图中每个灰度值的频数。可以看到，用频数最大的灰度值 Tmax 和由 Otsu 算法得到的 Totsu 重新计算阈值得到其值为 70 的 η，由于 Tmax 小于 220，根据公式 12 将比阈值 η 小的部分判别为背景，否则判别为前景。

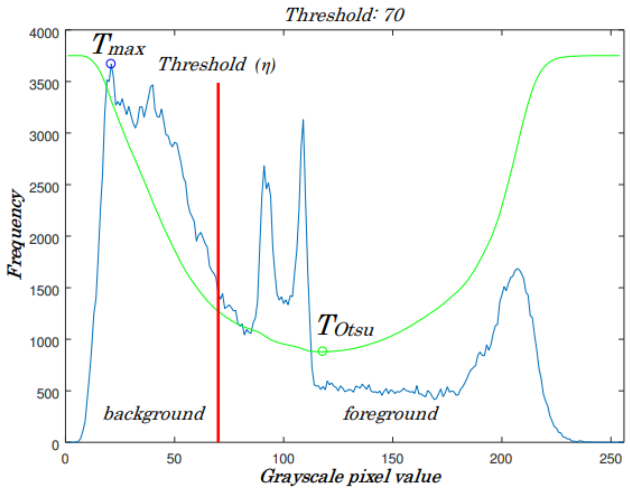


Fig.3. 图像灰度值分布图

K-means 聚类算法是一种常见的迭代求解的聚类分析算法，其步骤是，预将数据分为 K 组，则随机选取 K 个对象作为初始的聚类中心，然后计算每个对象与各个种子聚类中心之间的距离，把每个对象分配给距离它最近的聚类中心。聚类中心以及分配给它们的对象就代表一个聚类。每分配一个样本，聚类的聚类中心会根据聚类中现有的对象被重新计算。这个过程将不断重复直到满足某个终止条件。

本实验中的 K-means 聚类模型的 K 值为 3，也即将数据集划分为三个聚类，这三个聚类分别表示前景、背景和肤色区域。此模型输入为一个由图像的 Cb、Cr、H 分量和初步确

定的肤色区域组成的数据集。其中，初步确定的肤色区域是由对图像的前景进行实验原理1中提到的基于多颜色空间模型的肤色分割得到的。需要注意，构建数据集时需要对每个做归一化，这样可以保证四个分量对聚类结果的影响是等同的。

3. **基于帧差法的运动目标分割**

运动目标分割算法主要包括帧差法和背景差法。

背景差法是将当前帧与背景图像进行差分来得到运动目标区域。通过不含运动的帧来获取背景图像。常用的背景构建方法有时间平均法、单高斯法、混合高斯法等。但背景差分法结合肤色分割缺陷在于实验环境的实时性决定光照条件一直处于变动状态，背景差分中的背景更新问题较为复杂。因此，本实验中采取了帧差法。

帧差法是一种通过对视频图像序列中相邻两帧作差分运算来获得运动目标轮廓的方法，是一种基于运动图像序列中相邻帧图像具有较强的相关性而提出的检测方法。当监控场景中出现异常物体运动时，帧与帧之间会出现较为明显的差别，两帧相减，得到两帧图像亮度差的绝对值，判断它是否大于阈值来分析视频或图像序列的运动特性，确定图像序列中有无物体运动，从而实现运动目标的检测功能。

帧差法的优点是：算法实现简单，程序设计复杂度低；对光线等场景变化不太敏感，能够适应各种动态环境，稳定性较好。其缺点是：不能提取出对象的完整区域，只能提取出边界；同时依赖于选择的帧间时间间隔。对快速运动的物体，需要选择较小的时间间隔，如果选择不合适，当物体在前后两帧中没有重叠时，会被检测为两个分开的物体。

为了避免这些缺点对实验结果的影响，本实验中没有直接使用帧差法的结果，而把检测到的运动目标的质心作为判断是否为人手区域指标。假设人脸等肤色区域和背景中的类肤色区域相对稳定不动而只有人手在运动，当检测到有物体运动时，计算每个肤色区域到运动目标的质心的距离，取其中距离最小的肤色区域作为人手区域，则可以实现比较稳定而完整的人手检测。

另外，本实验中帧差法的阈值的选定参考了在 Hand Gesture Tracking System Using Adaptive Kalman Filter 一文中提出的公式：

$$\mathrm{T} = 0.05\bar{X} \quad (14)$$

其中 T 表示阈值，$\bar{X}$ 表示灰度图的平均灰度值。

4. 形态学处理

形态学处理是图像处理中应用最为广泛的技术之一，主要用于从图像中提取对表达和描绘区域形状有意义的图像分量，使后续的识别工作能够抓住目标对象最为本质的形状特征，如边界和连通区域等。同时像细化、像素化和修剪毛刺等技术也常应用于图像的预处理和后处理中，成为图像增强技术的有力补充。

结构元（Structuring Elements，SE）是类似于“滤波核”的元素，或者说类似于一

个“小窗”，在原图上进行“滑动”。结构元可以是任意大小和形状，一般由 0 和 1 的二值像素组成。结构元的原点相当于“小窗”的中心，其尺寸由具体的腐蚀或膨胀算子指定，结构元素的尺寸也决定着腐蚀或者膨胀的程度。

在经阈值处理提取出目标区域的二值图像之后，区域边缘可能并不理想，这时可以使用腐蚀或膨胀操作对区域进行收缩或扩张。腐蚀和膨胀是两种最基本也是最重要的形态学运算，它们是很多高级形态学处理的基础，很多其他的形态学算法都是由这两种基本运算复合而成。

腐蚀（Erosion）的原理是使用一个自定义的结构元，在二值图像上进行类似于“滤波”的滑动操作，然后将二值图像对应的像素点与结构元素的像素进行对比，得到的交集即为腐蚀后的图像像素。经过腐蚀操作，图像区域的边缘可能会变得平滑，区域的像素将会减少，相连的部分可能会断开。即使如此，各部分仍然属于同一个区域。

膨胀（Dilation）的原理是使用一个自定义的结构元素，在待处理的二值图像上进行滑动操作，然后将二值图像对应的像素点与结构元素的像素进行对比，得到的并集为膨胀后的图像像素。经过膨胀操作，图像区域的边缘可能会变得平滑，区域的像素将会增加，不相连的部分可能会连接起来，这些都与腐蚀操作正好相反。即使如此，原本不相连的区域任然属于各自的区域，不会因为像素重叠就发生合并。

开运算（Opening）的计算步骤是先腐蚀，后膨胀。通过腐蚀运算能去除小的非关健区域，也可以把离得很近的元素分隔开，再通过膨胀填补过度腐蚀留下的空隙。因此，通过开运算能去除孤立的、细小的点，平滑毛糙的边缘线，同时原区域面积也不会有明显的改变，类似于一种“去毛刺”的效果。

闭运算（Closing）的计算步骤与开运算正好相反，为先膨胀，后腐蚀。这两步操作能将看起来很接近的元素，如区域内部的空洞或外部孤立的点连接成一体，区域的外观和面积也不会有明显的改变。通俗地说，就是类似于“填空隙”的效果。与单独的膨胀操作不同的是，闭运算在填空隙的同时，不会使图像边缘轮廓加粗。

六、实验内容

1. 从电脑摄像头获取当前帧的 RGB 图像 img

MATLAB 里可以用 webcam 对象以及其属性 snapshot 从电脑摄像头获取 RGB 图像。右图为从 webcam 获取的原始图像。MATLAB 代码如下：

```
cam = webcam(1);
img = uint8(cam.snapshot);
```

Fig.4. 原始图像

2. 对 img 进行左右翻转以及双边滤波得到 filt_img（见 Fig.5）

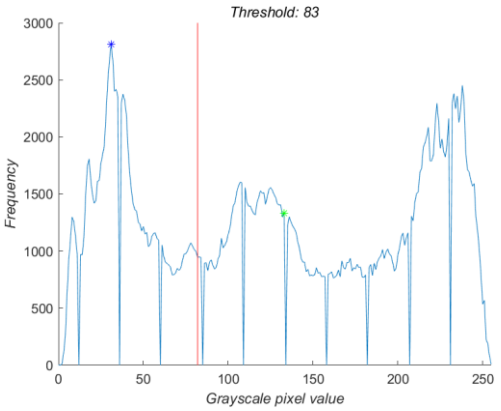
从 webcam 获取的图像不是镜面的，需要做左右翻转。此外，图像中会存在高斯白噪声，需要用滤波器去除噪声。本实验之所以使用双边滤波器，是因为高斯滤波、维纳滤波等降噪方法容易模糊图片的边缘细节，对于高频细节的保护效果并不明显。相比较而言，双边滤波器可以很好的边缘保护，即可以在去噪的同时，保护图像的边缘特性。代码实现如下：

```
img = flip(img,2);
filt_img = imbilatfilt (img);
```

Fig.5. 图像预处理

3. 根据 Otsu 阈值和 img 中频数最大的灰度值重新计算阈值 thres，对 filt_img 进行二值化处理得到 mask_hist

统计灰度值分布发现频数最大的灰度值 Tmax 为 32，Otsu 阈值 Totsu 为 134。由公式 11 计算阈值 thres 得到 83，因为 Tmax 小于 220，根据公式 12，下图中红线左侧为背景，右侧为前景。代码实现如下：




```
% RGB to Gray Scale
img_gray = rgb2gray(img);
% Get Otsu Threshold
Totsu = graythresh(img_gray)*255;
% Histogram
[n,~] = histcounts(img_gray,256);
[~,Tmax] = max(n);
% Calculate Threshold
if Tmax <= 10
    thres = round((Totsu+Tmax)/4);
else
    thres = round((Totsu+Tmax)/2);
end
% Binarize
mask_hist = uint8(zeros(size(img(:, :, 1))));
if Tmax <= 200
    mask_hist(img_gray > thres) = 1;
else
    mask_hist(img_gray < thres) = 1;
end
```

以下两个图分别为本实验使用的图像分割的结果和 Otsu 阈值分割的结果。可以看出，用 Otsu 阈值分割有时缺失前景信息比较严重，影响下一步肤色分割操作，而重新计算的阈值能够保留大部分包括肤色的前景信息，这就是不直接使用 Otsu 阈值的原因。

Fig.6. 图像分割

Fig.7. Otsu 阈值分割

4. 将 filt_img 转换到 YCbCr、HSV、归一化 RGB 颜色空间，获取 CbCr 分量、HS 分量和 Normalized RG 分量。

MATLAB 的 Image Processing Toolbox 中有将 RGB 图像转换到 YCbCr 和 HSV 颜色空间的函数 rgb2ycbcr 和 rgb2hsv。Normalized RG 分量的获取也简单，代码实现如下：

```
sumRGB = R+G+B;
normR = R ./ sumRGB;
normG = G ./ sumRGB;
```

5. 根据每个像素点的 CbCr 分量、HS 分量和 Normalized RG 分量，进行基于多颜色空间模型的肤色分割，得到 mask_skin。

根据公式 1、2、3、4、8，对图像的各分量做范围筛选处理，将满足所有范围的像素点判别为肤色，代码实现如下：

```
mask1 = zeros(n,m); mask1(Cb>77 & Cb<127) = 1;
mask2 = zeros(n,m); mask2(Cr>133 & Cr<173) = 1;
mask3 = zeros(n,m); mask3(H<=0.1 & 0.01<=H) = 1;
mask4 = zeros(n,m); mask4(S>=0.0754 & S<=0.6093) = 1;
mask8 = zeros(n,m); mask8((normR./normG)>1.185) = 1;
mask_skin = mask1 & mask2 & mask3 & mask4 & mask8;
```

Fig.8. 基于多颜色空间模型的肤色分割

6. 对 mask_hist 和 mask_skin 做逻辑与运算得到初步的肤色分割结果 mask。

Fig.9. mask

7. 构建数据集

先对图像的 Cb,Cr,H 分量和 mask 做归一化，然后构建数据集，得到行数为 n×m，列数为 4 的矩阵 dataset。其中，n 和 m 为图像的长和宽。代码实现如下：

```
h = H(:);
cr = Cr(:)/240;
cb = Cb(:)/240;
m = mask';
m = double(m(:));
dataset = [cr,cb,h,m];
```

dataset

307200x4 double

	1	2	3	4
1	0.5708	0.4917	0.0762	0
2	0.5708	0.4917	0.0787	0
3	0.5708	0.4917	0.0897	0
4	0.5708	0.4958	0.0992	0
5	0.5708	0.4958	0.1087	0
6	0.5708	0.4958	0.1122	0
7	0.5708	0.4958	0.1154	0
8	0.5667	0.4958	0.1195	0
9	0.5667	0.4958	0.1212	0
10	0.5667	0.4917	0.1204	0
11	0.5625	0.4875	0.1186	0
12	0.5625	0.4833	0.1133	0
13	0.5625	0.4792	0.1087	0
14	0.5583	0.4750	0.1023	0

Fig.10. 构建数据集

8. 用 dataset 进行 K 均值聚类的训练或测试，得到最终的肤色分割结果 final_skin。

MATLAB 的 Statistics and Machine Learning Toolbox 里有 kmeans 函数执行 k 均值聚类，以将(n×m)×4 数据矩阵 dataset 的观测值划分为 3 个聚类，并返回包含每个观测值的簇索引的(n×m)×1 向量 idx 和 k 个聚类质心位置。MATLAB 代码如下：

```
[idx,C]=kmeans(dataset,3,'MaxIter',100,'Distance','cityblock','Replicates',5,'Display','final');
```

由于训练结果中每个标注所代表的意义是不确定的，也就是说不知道哪一个标注表示背景、前景或肤色区域（见 Fig.10）。因此，需要进行对聚类的显示的标注（见 Fig.11）。Fig.12 中，白色为肤色区域，灰色为前景，黑色为背景。

Fig.11. Kmeans 训练

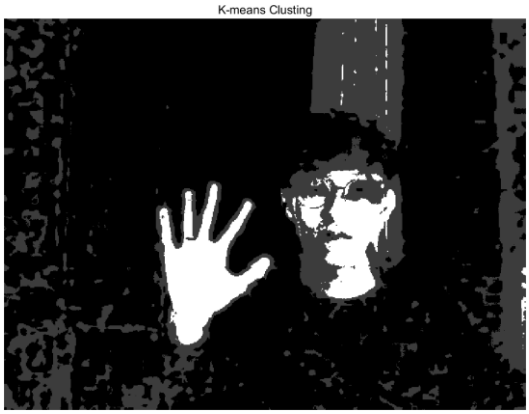


Fig.12. 数据标注

为了节省设备上的内存并减少运行时间，本实验使用了 kmeans 和 pdist2 来分离训练和测试。K-means 聚类算法的测试直接使用在训练中获得的每个聚类的质心对测试数据集进行分类，使用 pdist2 函数找到距离每个测试数据点最近的质心。代码实现如下：

```
[~,idx] = pdist2(C,dataset,'euclidean','Smallest',1);
```

9. 对 final_skin 进行二值化后进行形态学处理得到 skin_mask。

图像二值化后发现，所得到的边界不平滑，肤色区域具有一些判错，背景区域上则散布着一些小的噪声。本实验首先使用了以小正方形为结构元的开运算，去除孤立、细小的毛刺，然后用闭运算将区域内部的空洞或外部孤立的点连接成一体。因为有时手关节部分因细小的色度差判别为非肤色区域，所以用竖轴更长的十字形结构元做了膨胀操作，将人手区域中分离的部分相连接。最后用以菱形为结构元的腐蚀操作，将图像区域的边缘变得平滑。

结构元的定义以及四种形态学处理算子可以用 Image Processing Toolbox 的一些函数实现：strel 函数可以定义各种形状和大小的结构元，imopen、imclose、imerode 和 imdilate 分别是开运算、闭运算、腐蚀和膨胀操作。代码实现如下：

```
% Binarize
bw = final_skin == 255;
% Morphological Image Processing
se = strel('square',3);
se90 = strel('line',10,90);
se0 = strel('line',3,0);
seD = strel('diamond',1);
bw = bwareaopen(bw,200);
bw = imopen(bw,se);
bw = imclose(bw,se);
bw = imdilate(bw,[se90 se0]);
bw = imfill(bw,'holes');
bw = imerode(bw,seD);
```

Fig.13. 形态学处理

10. 用当前帧 img 和上一帧 prev_img 做帧差法，检测运动目标，得到 mask_md。

由于类肤色区域的细小的运动可能影响运动目标的检测，需要进行一些形态学处理。代码实现如下：

```
% Frame Difference
Fdt = abs(img - prev_img);
% rgb2gray
Fdg = 0.299*Fdt(:, :, 1) + 0.587*Fdt(:, :, 2) + 0.114* Fdt(:, :, 3);
% calculate T
X = mean(Fdg, 'all');
T = 0.05 * X;
Q = imbinarize(Fdg,T); % D(t,t+1)
% Morphological Image Processing
se = strel('square',5);
bw = imopen(Q,se);
bw = imclose(bw,se);
bw = imfill(bw,'holes');
bw = imclearborder(bw,4);
bw = bwareaopen(bw,300);
```

11. 计算 mask_md 中每个连通区域的质心 moveC，再计算整个二值化图像的质心 moveC'。

具体实现方法是，使用 regionprops 的 Centroid 属性获取每个连通区域的质心，用这些质心构造一个多边形，计算此多边形的质心作为整个二值化图像的质心。为了保证人手不动情况下也能正常判断人手区域，质心 moveC'的更新只在检测到运动目标时发生。Fig.14 中，白色连同区域是通过帧差法获得的运动的人手区域，红色星号代表每个连通区域的质心，黄色星号代表整个二值化图像的质心。代码实现如下：

```
stat = regionprops(mask_md, 'centroid');
moveC = cat(1,stat.Centroid);
if ~isempty(moveC)
    moveC_x = moveC(:,1);
    moveC_y = moveC(:,2);
    % Points convert to Polygon
    pgon = polyshape(moveC_x,moveC_y);
    % Calculate Centroid of Polygon
    [moveCC_x,moveCC_y] = centroid(pgon);
end
```

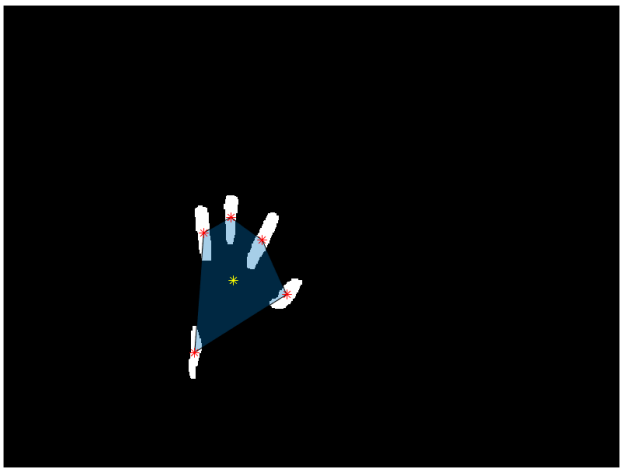


Fig. 14. 运动目标以及其质心

12. 计算 skin_mask 中每个连通区域的质心 skinC，再计算每个质心到 moveC'的距离，并将其距离最小的连同区域作为人手区域 hand_region。

Fig. 15 中，白色区域为 skin_mask 中的肤色区域，红色星号表示每个连同区域的质心 skinC，蓝色小圆代表运动目标的质心 moveC'。可以看到，最左侧的肤色区域离运动目标的质心最近，可以判断为人手区域（见 Fig. 16）。代码实现如下：

```
% Select Hand Region
labeledImage = bwlabel(skin_mask);
stats = regionprops(labeledImage, 'Centroid','BoundingBox');
if ~isempty(stats)
   % Calculate Distance
   skinC = vertcat(stats.Centroid);
   skinC_x = skinC(:, 1);
   skinC_y = skinC(:, 2);
   distances = sqrt((moveCC_x-skinC_x).^ 2+(moveCC_y-skinC_y).^ 2);
   % Find Nearest
   [~, idx] = min(distances);
   hand_region = ismember(labeledImage, idx);
end
```

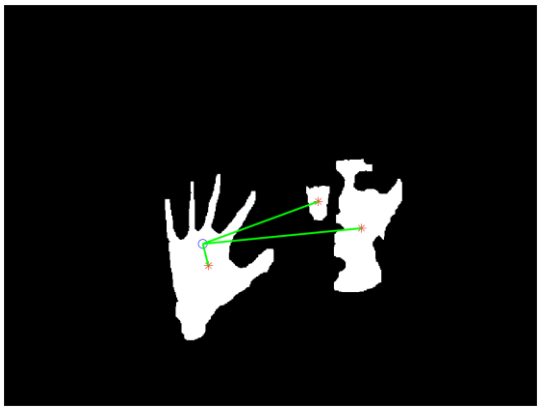


Fig. 15. 计算质心间距离

Fig. 16. 确定人手区域

13. 基于上一步骤确定的人手区域在原始图像的上画红色边框。

Fig.17. 最终结果

14. GUI 设计

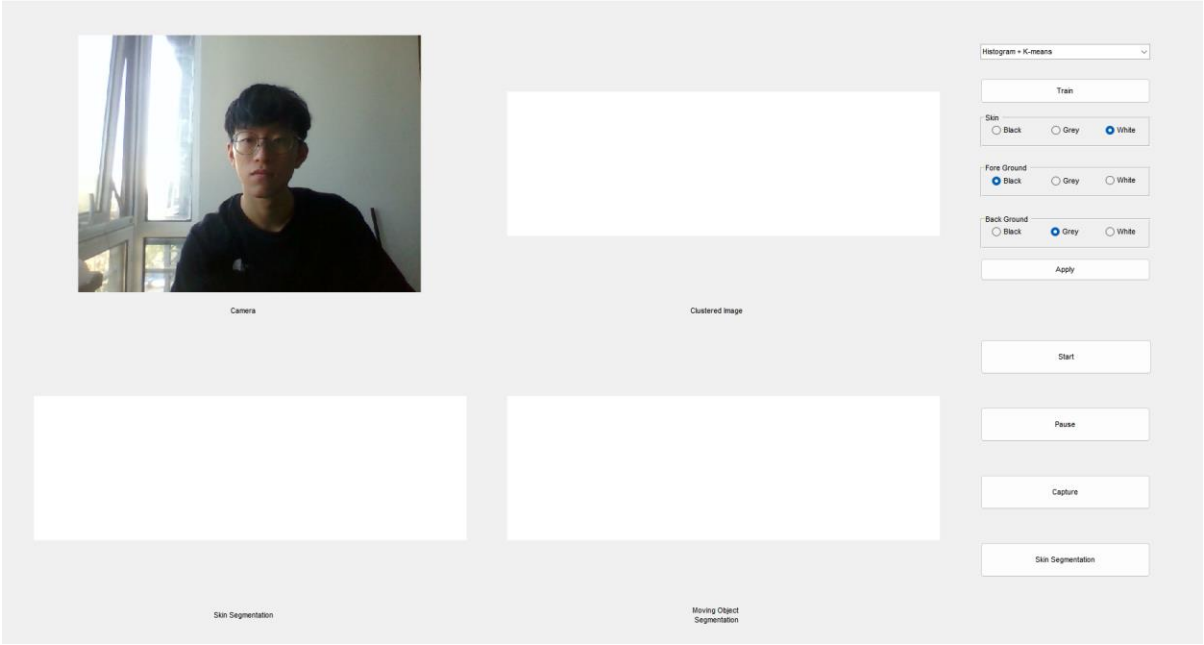


Fig.18. GUI 设计

A. axes

GUI 包括四个 axes，分别是 cam_axes（左上）、train_axes（右上）、skin_axes（左下）和 move_axes（右下）。cam_axes 用于显示从电脑摄像头获取的图像或最终人手识别的结果；train_axes 中显示 K-means 聚类的训练结果、聚类标注结果或人手区域的二值化图像 mask_skin。skin_axes 显示各种肤色分割算法的结果 final_skin。move_axes 中显示运动目标分割的结果、各连通区域的质心和整个运动目标的质心（参见 Fig.14）。

B. mode_menu

在此 popupmenu 中可以选择各种肤色分割算法进行人手识别。其默认值为基于灰度值分布和 K-means 聚类的肤色分割算法。

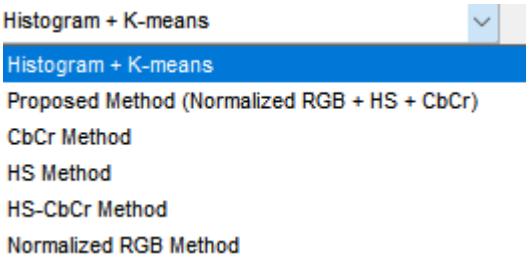


Fig.19. mode_menu

C. K-means 聚类训练

当用户 mode_menu 中选择“Histogram + K-means”时，会显示如下界面。

Train 按键：进行 K-means 聚类的训练并在 train_axes 中显示对聚类随机标注的训练结果（参见 Fig.11）。

Skin、Foreground、Background 按键：可以用这些按键显示地对聚类标注。例如，若训练结果中肤色区域随机标注为黑色，则选择 Skin 中的 Black 按键表示训练结果中黑色部分为肤色区域。

Apply 按键：确定各聚类表示的意义并将结果显示到 train_axes 中（参见 Fig.12）。其中，白色表示肤色区域，灰色表示前景，黑色表示背景。

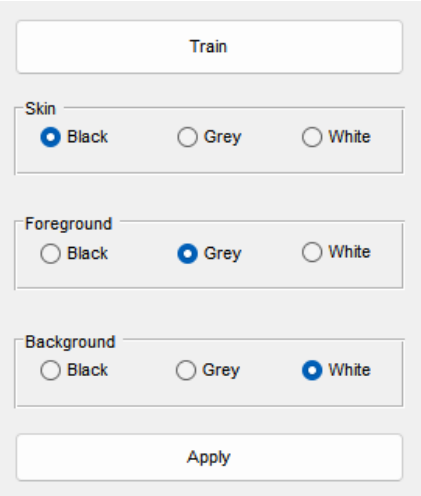


Fig.20. 训练和数据标注

D. Buttons

GUI 界面的左下方有四个按键 Start、Pause、Capture 和 Skin Segmentation。

Start 按键：进入无限循环，开始人手检测并在各个 axes 中实时显示各步骤的结果。

Pause 按键：停止无限循环。

Capture：保存当前图像。

Skin Segmentation：对当前图像进行各种肤色分割算法并显示在一起。

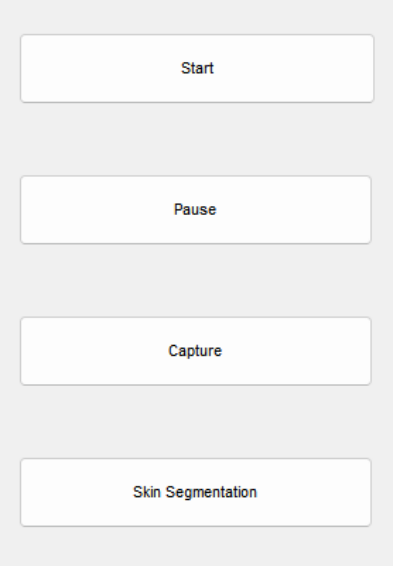


Fig.21. Buttons

七、 实验结果和分析

1. 肤色分割结果分析

肤色分割的结果会直接影响最终的人手识别。即是运动目标分割可以准确地确定哪一个肤色块是人手区域，若肤色分割的结果不可靠，则得到的人手区域也不会很准确，与实际结果不一致的概率非常高。例如，如果某个肤色分割算法将肤色像素点判别为非肤色的概率高，则得到的人手区域往往比实际的人手区域小；如果某个肤色分割算法将非肤色像素点判别为肤色的概率高，则得到的人手区域可能比实际人手区域大。而对本实验采用的基于灰度值分布和 K-means 聚类的肤色分割模型（Histogram + Kmeans 模型）的测试结果显示，此模型无论在包含类肤色区域的复杂环境或单一背景中，能够有效地去除非肤色区域并很准确地确定肤色区域。以下测试中会进行基于归一化 RGB、HS、CbCr、HS-CbCr 颜色空间的肤色分割、Proposed Method[2]以及 Histogram + Kmeans 肤色分割，并用下列评价指标对其结果进行客观分析。

A. 评价指标

本测试中使用了准确率（Accuracy，A）、精准率（Precision，P）、召回率（Recall，R）以及误检率（False Positive Rate，FPR）来评价各种肤色分割模型的结果。准确率表示预测结果中正确的占总预测值的比例；精准率代表预测结果中某类别预测正确的概率；召回率表示真实值中某类别被预测正确的概率；误检率为将负类预测为正类的概率。其计算公式如下：

$$\mathrm{A} = \frac{TP+TN}{TP+TN+FP+FN} \qquad (15)$$

$$\mathrm{P} = \frac{TP}{TP+FP} \qquad (16)$$

$$\mathrm{R} = \frac{TP}{TP+FN} \qquad (17)$$

$$\mathrm{FNR} = \frac{FN}{TN+FP} \qquad (18)$$

其中，TP（True Positive）为将正类预测为正类的样本数，FP（False Positive）为将负类预测为正类的样本数，FN（False Negative）为将正类预测为负类的样本数，TN（True Negative）为将负类预测为负类的样本数。

B. 测试 1

测试 1 中肤色分割的输入为在黑色背景中只有人手的图像，Fig.22 为各种肤色分割算法的结果。其中，第二各图像为实际肤色区域，最后一个图为对基于灰度值分布和 K-means 聚类的肤色分割结果进行形态学处理后得到的二值化图像。可以发现，基于 HS、CbCr 和 HS-CbCr 颜色空间上进行的肤色分割的结果有很高的精准率、准确率和召回率；Proposed Method 和 Histogram + K-means 模型的分割结果误检率非常低，而其精准率不如其他肤色分割算法高，但通过形态学处理可以有效地提高其精准率，得到比较完整的肤色区域。

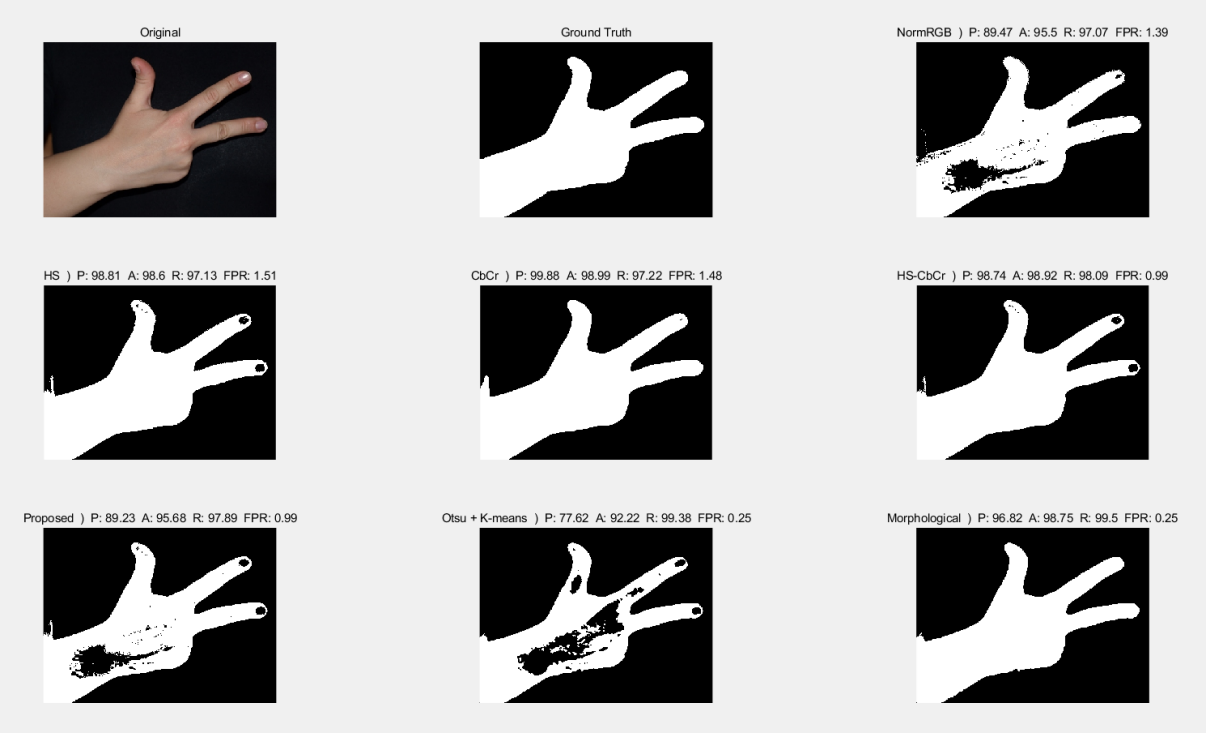


Fig.22. 测试 1

C. 测试 2

测试 2 中肤色分割的输入图像中背景很容易被判别成肤色。因此，除了基于归一化 RGB 颜色空间的肤色分割和 Histogram + K-means 模型，其他肤色分割算法均将背景中的非肤色区域判别为肤色，有很高的误检率。此外，对 Histogram + K-means 模型做形态学处理后有 96.66%的准确率和 4.65%的误检率，其结果是比较可靠的。

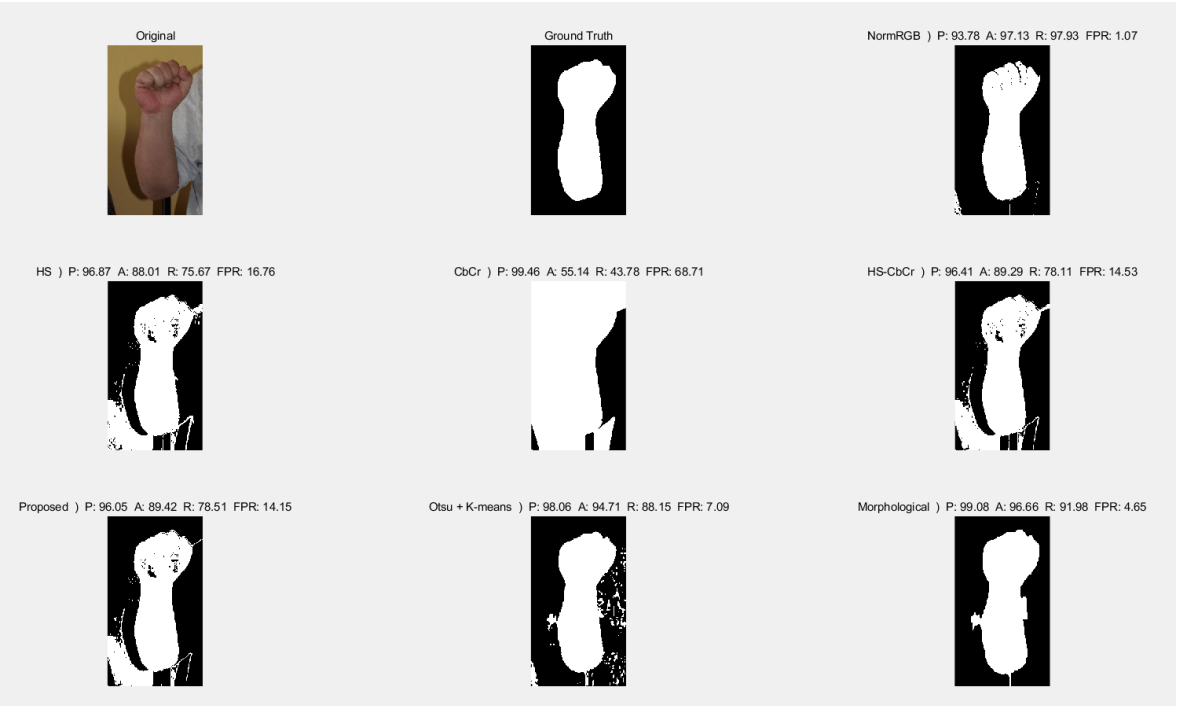


Fig.23. 测试 2

D. 测试 3

因为大部分情况下从电脑摄像头获取的图像中会包括人脸等肤色区域，所以也测试了对整个人的肤色分割。从 Fig.14 可以看出，除了 Histogram + K-means 模型，其他肤色分割模型没有准确地判断肤色区域。此原因是论文给出的各个颜色分量的阈值范围很可能是针对于白种人和黄种人，而图像中人的肤色比较暗，肤色像素点的各个分量均不在指定范围内。这是对各种色彩分量做单纯的范围筛选来判断是否为肤色的缺陷。而 Histogram + K-means 模型是先进行初步的肤色分割后，用 K-means 聚类算法对图像的 CbCr 和 H 分量聚类，再次判断肤色区域，因此，不管肤色分割的对象的人种，此模型能够比较准确地判断肤色区域。

Fig.24. 测试 3

2. 实验结果

实验结果发现，在非肤色背景下六个算法的人手检测结果都很可靠，均有很高的准确率和很低的误检率，而在复杂环境中，基于 CbCr 和 HS 颜色空间模型的人手检测的结果不太可靠，虽然基于帧差法的运动目标分割准确地确定了人手区域，但因为这些肤色分割算法没有有效地去除背景中的类肤色区域，其人手区域对应的连同区域也包含了背景中的部类肤色像素点，从而得到了比实际人手区域更大的人手区域。此外，基于帧差法的最小质心间距离判断人手区域的的结果也非常可靠，适当的阈值处理和形态学处理有效地去除了前景中类肤色物体的微小运动可能带来的干扰。因为运动目标质心的更新只在检测到运动目标时发生，无论人手在运动或固定不动，能够正确判断哪一个连通区域为人手区域。

A. K-means 聚类训练和聚类标注

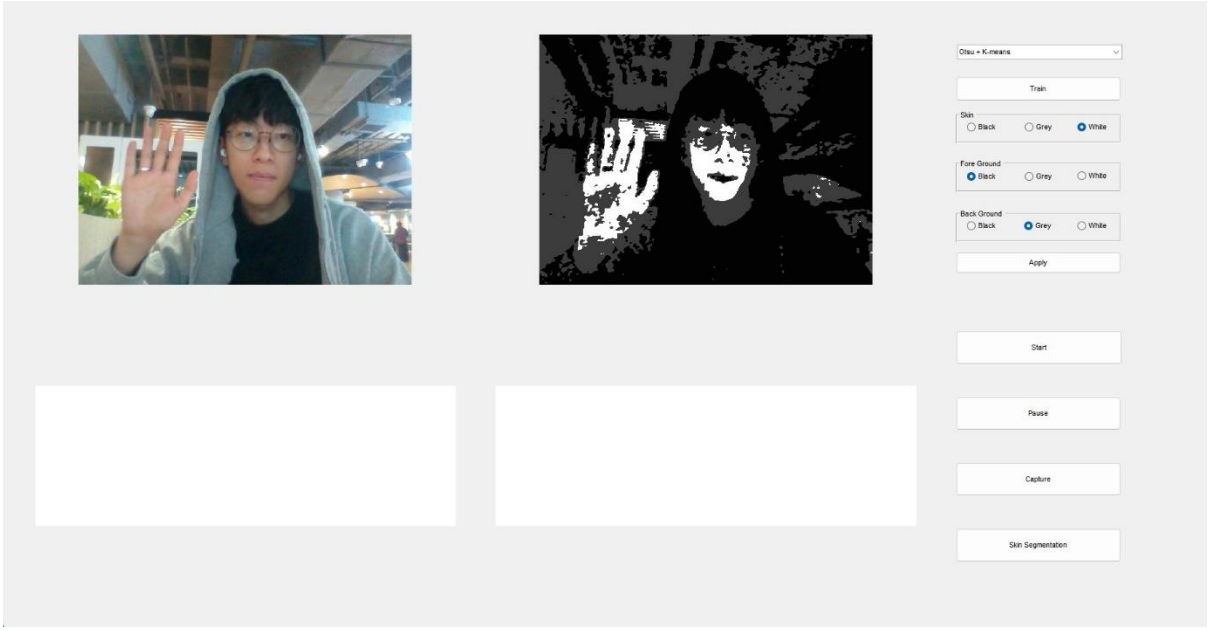


Fig.25. K-means 聚类训练结果

B. Histogram + K-means 模型

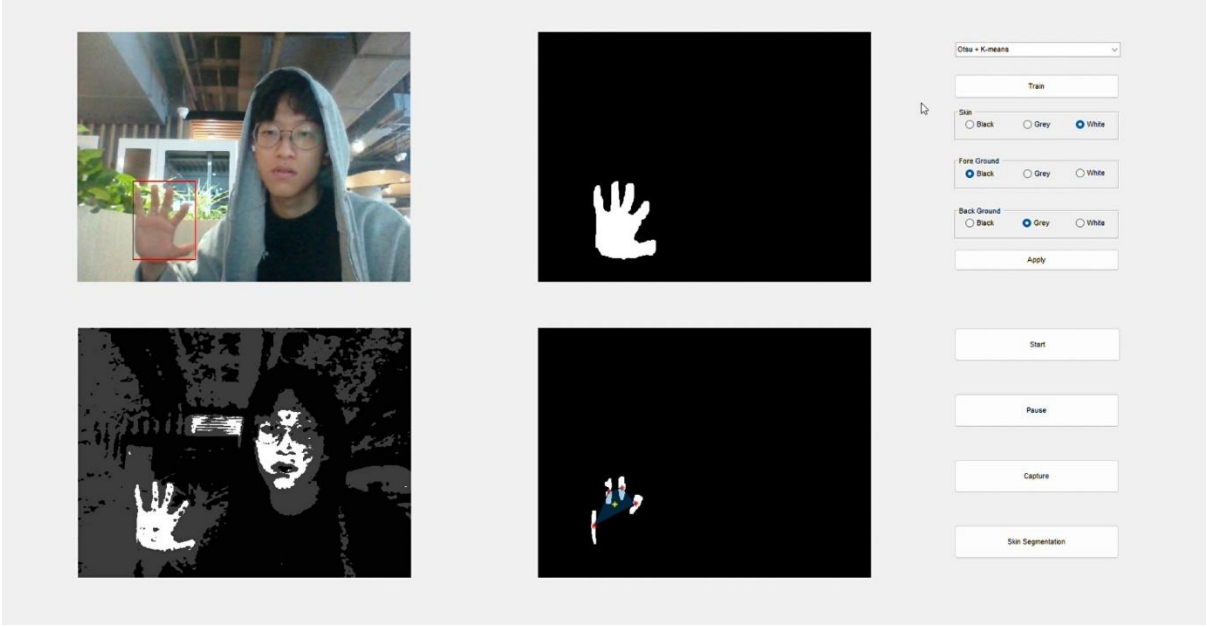


Fig.26. 基于 Histogram + K-means 模型的人手识别

C. Proposed Method

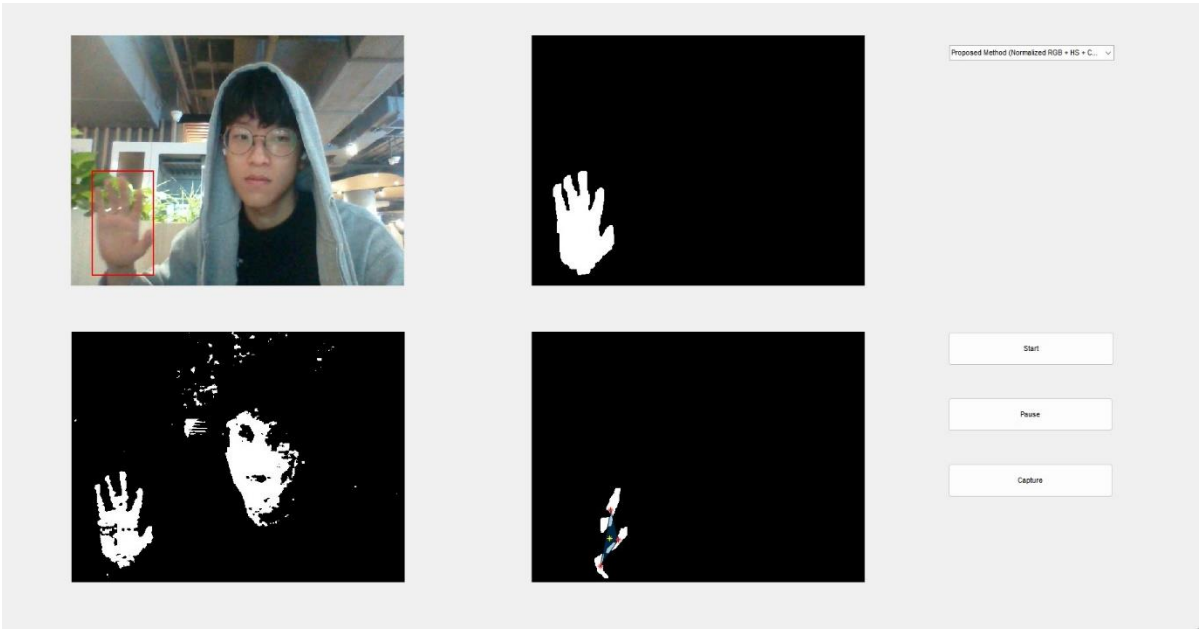


Fig.27. 基于 Proposed Method 的人手识别

D. 基于 CbCr 颜色空间的肤色分割

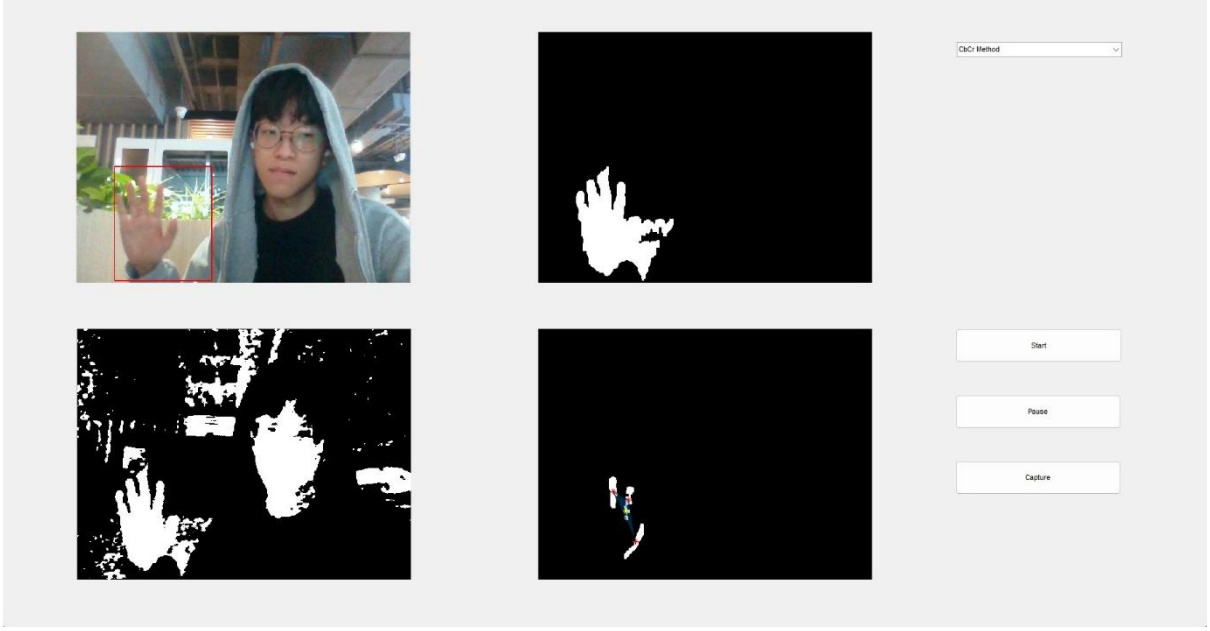


Fig.28. 基于 CbCr 颜色空间模型的人手识别

E. 基于 HS 颜色空间的肤色分割

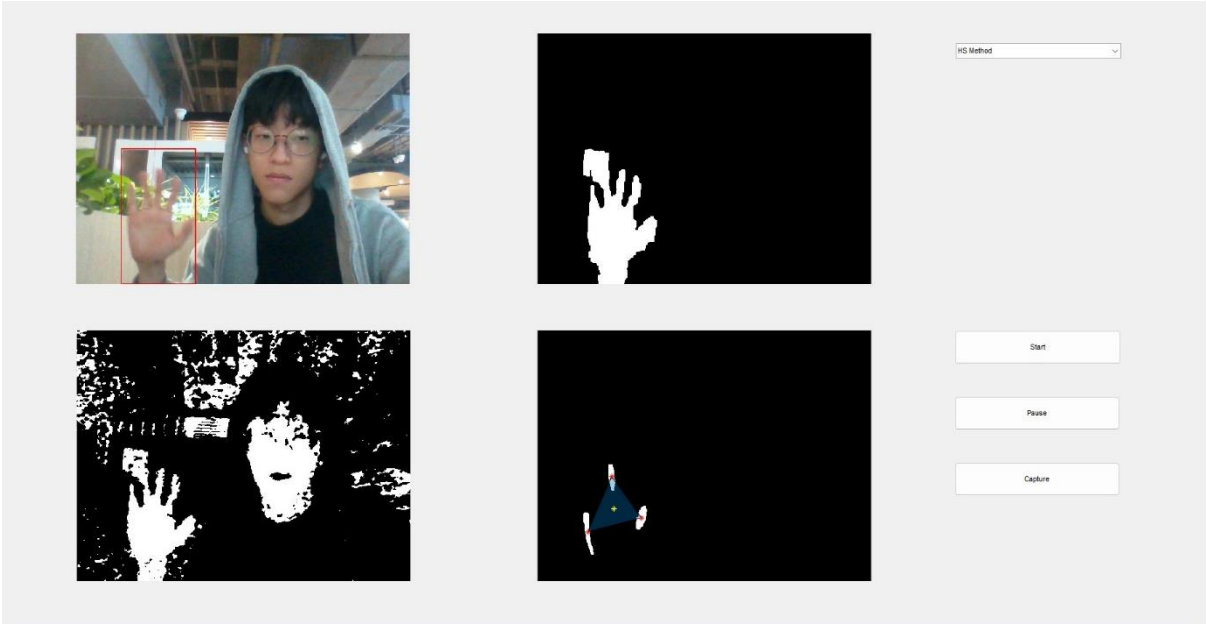


Fig.29. 基于 HS 颜色空间模型的人手识别

F. 基于 HS-CbCr 颜色空间的肤色分割

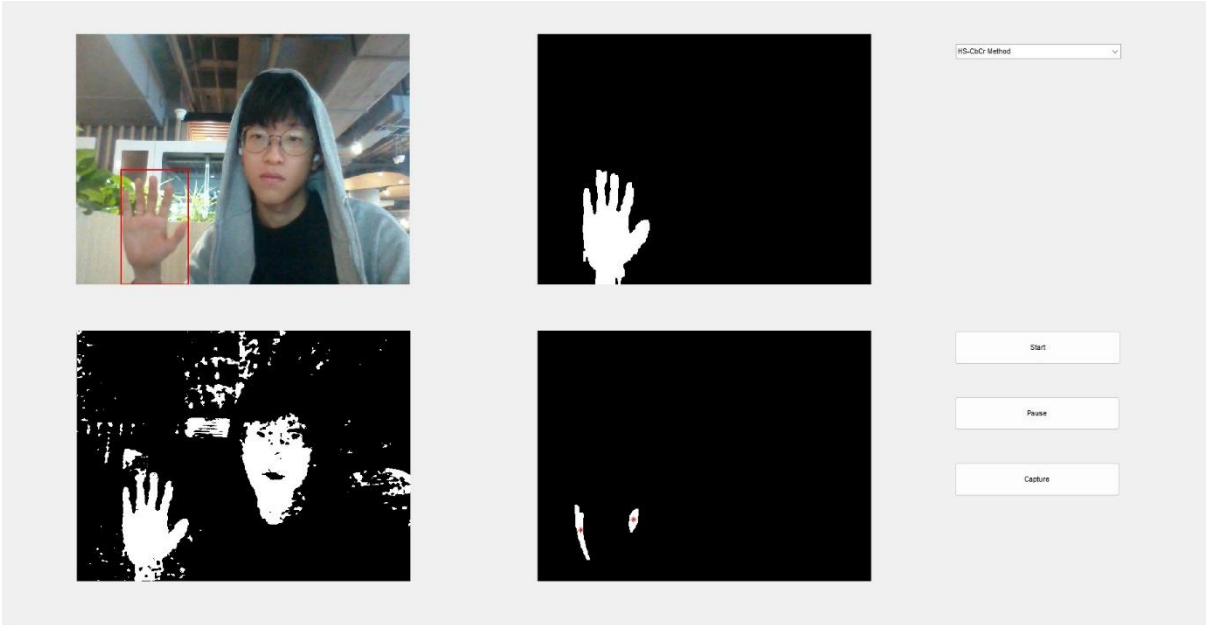


Fig.30. 基于 HS-CBCr 颜色空间模型的人手识别

G. 基于归一化 RGB 颜色空间的肤色分割

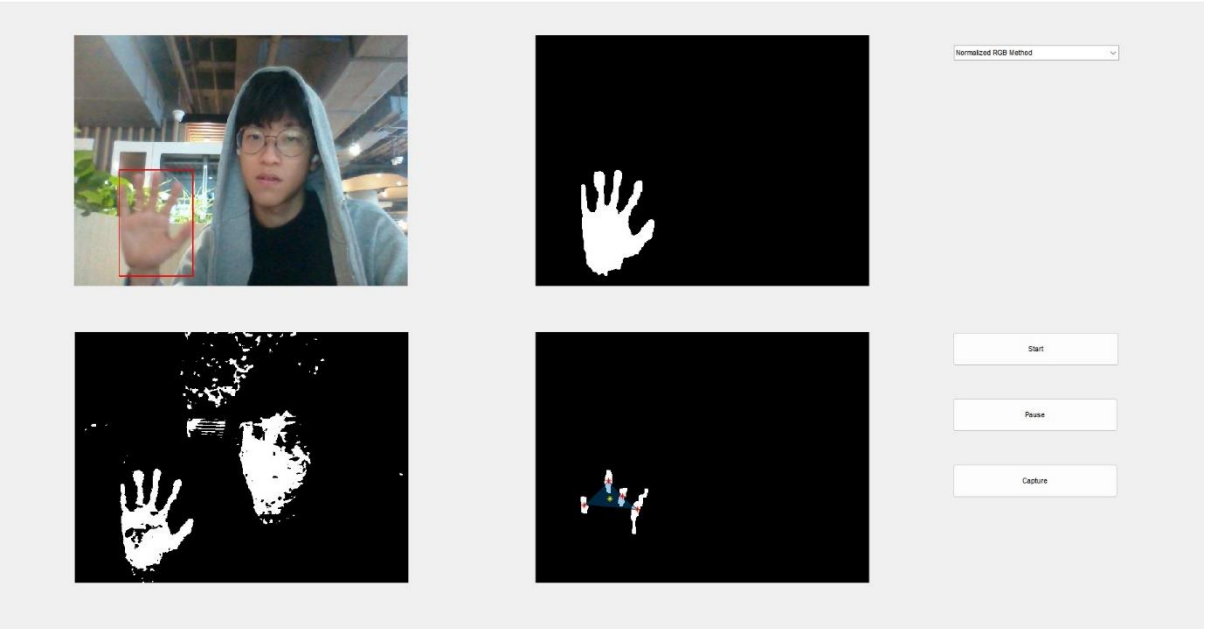


Fig.31. 基于归一化 RGB 颜色空间模型的人手识别

八、讨论

本实验中实现的人手识别方法在特定条件下是非常准确、可靠的，即只有一个手在运动而其它物体稳定不动的情况下，其对人手区域的判断是准确的。这一特性体现了本算法的局限性：如果背景中有物体激烈运动，则得到的结果很可能不是人手区域；图像中有双手在运动时，其运动目标的质心会处于图像中心，从而将人脸判断为人手区域。能够解决此问题的方法有使用 ResNet 等神经网络对肤色区域进行模板匹配，判断哪几个肤色区域为人手区域。另外，当从摄像头获取的图像很大时，K-means 聚类的训练和测试所需要的时间会很长，会影响识别的实时性。

九、参考文献

[1] Jure Kovac, Peter Peer, and Franc Solina. Human Skin Colour Clustering for Face Detection. 2003

[2] Rahmat R.F., Chairunnisa T., Gunawan D., & Sitompul O.S. Skin color segmentation using multi-color space threshold. 2016

[3] Emir Buza, Amila Akagic, Samir Omanovic. Skin Detection Based on Image Color Segmentation with Histogram and K-Means Clustering. 2017

[4] Mohd Shahrimie, Shahrel Azmin Suandi. Hand Gesture Tracking System Using Adaptive Kalman Filter. 2010

十、代码

1. FinalLab.m

```
function varargout = FinalLab(varargin)

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @FinalLab_OpeningFcn, ...
                   'gui_OutputFcn',  @FinalLab_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before FinalLab is made visible.
function FinalLab_OpeningFcn(hObject, eventdata, handles, varargin)
global cam;
global init_img;
set(hObject,'toolbar','figure');
cam = webcam(1);
warning off;
init_img = uint8(cam.snapshot);
init_img = flip(init_img,2);
axes(handles.cam_axes);
imshow(init_img);

axes(handles.train_axes);
imshow(uint8([255 255 255]));

axes(handles.skin_axes);
imshow(uint8([255 255 255]));

axes(handles.move_axes);
imshow(uint8([255 255 255]));

handles.output = hObject;

% Update handles structure
guidata(hObject, handles);


% --- Outputs from this function are returned to the command line.
function varargout = FinalLab_OutputFcn(hObject, eventdata, handles)

varargout{1} = handles.output;
clear;

% --- Executes on selection change in mode_menu.
function mode_menu_Callback(hObject, eventdata, handles)
global kmeans;
global mode;
kmeans = 0;
```

```
mode_val = get(handles.mode_menu,'Value');
if mode_val == 1
    kmeans = 1;
    mode = 'Proposed';%'K-means';
% elseif mode_val == 2
%     kmeans = 1;
%     mode = 'CbCr';
elseif mode_val == 2
    mode = 'Proposed';
elseif mode_val == 3
    mode = 'CbCr';
elseif mode_val == 4
    mode = 'HS';
elseif mode_val == 5
    mode = 'HS-CbCr';
elseif mode_val == 6
    mode = 'NormRGB';
else
    kmeans = 1;
    mode = 'Proposed';
end
if kmeans == 1
    set(handles.train_button,'visible','on');
    set(handles.apply_button,'visible','on');
    set(handles.skin_button,'visible','on');
    set(handles.skinSeg_button,'visible','on');
    set(handles.foreGround_button,'visible','on');
    set(handles.backGround_button,'visible','on');
else
    set(handles.train_button,'visible','off');
    set(handles.apply_button,'visible','off');
    set(handles.skinSeg_button,'visible','off');
    set(handles.skin_button,'visible','off');
    set(handles.foreGround_button,'visible','off');
    set(handles.backGround_button,'visible','off');
end

% --- Executes during object creation, after setting all properties.
function mode_menu_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
global mode
global kmeans

kmeans = 1;
mode = 'Proposed';



% --- Executes on button press in train_button.
function train_button_Callback(hObject, eventdata, handles)
global init_img;
global mode;
global n;
global m;
global C;
global idx;
global cam;
set(handles.text2,'String','Clustered Image')
init_img = cam.snapshot;
init_img = flip(init_img,2);
imshow(init_img, 'Parent', handles.cam_axes);
```

```
[idx,C] = trainSkin(init_img,mode);
[n,m,~] = size(init_img);
res = uint8(zeros(n,m));
i = 1;
for x = 1:n
    for y = 1:m
        if idx(i) == 3
            res(x,y,:) = 255;
        elseif idx(i) == 2
            res(x,y,:) = 60;
        else
            res(x,y,:) = 0;
        end
        i = i + 1;
    end
end
res = medfilt2(res);
axes(handles.train_axes)
imshow(res)


% --- Executes on button press in apply_button.
function apply_button_Callback(hObject, eventdata, handles)
global idx;
global n;
global m;
global skin;
global bg;
global fg;
global train_mask;
val = get(handles.skin_button,'SelectedObject');
skin_val = get(val,'string');
val = get(handles.foreGround_button,'SelectedObject');
fg_val = get(val,'string');
val = get(handles.skin_button,'SelectedObject');
bg_val = get(val,'string');
if skin_val == "Black"
    skin = 1;
elseif skin_val == "Grey"
    skin = 2;
else
    skin = 3;
end
if fg_val == "Black"
    fg = 1;
elseif fg_val == "Grey"
    fg = 2;
else
    fg = 3;
end
if bg_val == "Black"
    bg = 1;
elseif bg_val == "Grey"
    bg = 2;
else
    bg = 3;
end
train_mask = cluster2mask(idx,n,m,skin,bg,fg);
train_mask = medfilt2(train_mask);
axes(handles.train_axes)
imshow(train_mask)


% --- Executes on button press in pause_button.
function pause_button_Callback(hObject, eventdata, handles)
handles.stop_now = 1;
```

```
guidata(hObject, handles); % Update handl



% --- Executes on button press in start_button.
function start_button_Callback(hObject, eventdata, handles)
global cam;
global mode;
global kmeans;
global C;
global n;
global m;
global skin;
global bg;
global fg;
global init_img;
handles.stop_now = 0;
guidata(hObject,handles);
set(handles.text2,'String','Hand Region')

% filt = 1/9*ones(3);
se = strel('square',3);
se90 = strel('line',10,90);
% se45 = strel('line',8,45);
se0 = strel('line',3,0);
seD = strel('diamond',1);

% Get Previous Frame
img = uint8(cam.snapshot);
img = flip(img,2);
prev_img = imbilatfilt(img);
% prev_img = imfilter(img,filt);

% Initialize Centroid of Hand (Center of Image)
[n,m,~] = size(img);
moveCC_x = n/2;
moveCC_y = m/2;

while ~(handles.stop_now)
    % Get Current Frame
    img = uint8(cam.snapshot);
    img = flip(img,2);
    filt_img = imbilatfilt(img);

    % Skin Detection
    if kmeans == 1
        [idx] = testSkin(filt_img,C,mode);
        final_skin = cluster2mask(idx,n,m,skin,bg,fg);
    else
        final_skin = skinSeg(filt_img,mode);
    end

    % Motion Detection
    mask_md = motionDetection(filt_img,prev_img);
    imshow(mask_md, 'Parent', handles.move_axes);

    % Centroid of Motion Detection BW
    stat = regionprops(mask_md, 'centroid');
    moveC = cat(1,stat.Centroid);
    if ~isempty(moveC)
        moveC_x = moveC(:,1);
        moveC_y = moveC(:,2);
        % Points convert to Polygon
        pgon = polyshape(moveC_x,moveC_y);
```

```
        % Calculate Centroid of Polygon
        [tmpx,tmpy] = centroid(pgon);
        if ~(isnan(tmpx)||isnan(tmpy))
            moveCC_x = tmpx;
            moveCC_y = tmpy;
        end
        hold(handles.move_axes,'on');
        plot(pgon,'Parent', handles.move_axes);
        plot(moveCC_x,moveCC_y,'y*','Parent', handles.move_axes)
        plot(moveC_x,moveC_y,'r*','Parent', handles.move_axes)
        hold(handles.move_axes,'off');
    end

    % Binarize
    bw = final_skin > 100;

    % Morphological Image Processing
    bw = bwareaopen(bw,200);
    bw = imopen(bw,se);
    bw = imclose(bw,se);
    bw = imdilate(bw,[se90 se0]);
%     bw = imdilate(bw,se45);
    bw = imfill(bw,'holes');
    skin_mask = imerode(bw,seD);

    % Select Biggest Three Blob
    skin_mask = bwareaopen(bwareafilt(skin_mask,3),5000);

    % Select Hand Region
    labeledImage = bwlabel(skin_mask);
    stats = regionprops(labeledImage, 'Centroid','BoundingBox');
    if ~isempty(stats)
        % Calculate Distance
        skinC = vertcat(stats.Centroid);
        skinC_x = skinC(:, 1);
        skinC_y = skinC(:, 2);
        distances = sqrt((moveCC_x - skinC_x) .^ 2 + (moveCC_y - skinC_y) .^ 2);

        % Find Nearest
        [~, idx] = min(distances);
        hand_region = ismember(labeledImage, idx);

        % Draw ROI
        handRegion = stats(idx).BoundingBox;
        final =
insertShape(img,'Rectangle',[handRegion(1),handRegion(2),handRegion(3),handRegi
on(4)],'LineWidth',2,'Color','red');
        imshow(final,'Parent', handles.cam_axes);
        imshow(hand_region, 'Parent', handles.train_axes);
    else
        imshow(img, 'Parent', handles.cam_axes);
        imshow(skin_mask, 'Parent', handles.train_axes);
    end
    imshow(final_skin, 'Parent', handles.skin_axes);
    drawnow;

    % Update Data
    handles = guidata(hObject);

    % Update Previous Frame
    prev_img = filt_img;

end
init_img = img;
moveCC_x
```

moveCC_y

```
% --- Executes on button press in capture_button.
function capture_button_Callback(hObject, eventdata, handles)
global init_img;
global cam;
% init_img = cam.snapshot;
% init_img = flip(init_img,2);
imshow(init_img, 'Parent', handles.cam_axes);
t = datetime('now');
tstr = strcat(datestr(t),'.jpg');
tstr = strrep(tstr,':',';')
imwrite(init_img,tstr);


% --- Executes on button press in skinSeg_button.
function skinSeg_button_Callback(hObject, eventdata, handles)
global init_img;
global train_mask;
img = init_img;
figure;
subplot(3,3,1); imshow(img); title("Original")
m = ["NormRGB","HS","CbCr","HS-CbCr","Proposed"];
for i=2:6
    subplot(3,3,i)
    skinMask = skinSeg(img,m(i-1));
    mask = skinMask >= 1;
    imshow(mask)
    title(m(i-1))
end
subplot(337)
imshow(train_mask);
t = "Histogram + K-means";
title(t)
subplot(338);
bw = train_mask > 200;
imshow(bw);
title("Binarized Image");
subplot(339)
se = strel('square',3);
se90 = strel('line',10,90);
se0 = strel('line',3,0);
seD = strel('diamond',1);
bw = bwareaopen(bw,200);
bw = imopen(bw,se);
bw = imclose(bw,se);
bw = imdilate(bw,[se90 se0]);
bw = imfill(bw,'holes');
bw = imerode(bw,seD);

imshow(bw);
title("Morphological Image")
```

2. trainSkin.m

```
function [idx,C] = trainSkin(img,mode)
% Input
% img: RGB Image
% mode: 'Proposed'

% Output
% idx: Index of Clustered Image
% C: Centroid of Clusters
```

```
% RGB to Gray Scale
img_gray = rgb2gray(img);
% Get Otsu Threshold
Totsu = graythresh(img_gray)*255;
% Histogram
[n,~] = histcounts(img_gray,256);
[~,Tmax] = max(n);
% Calculate Threshold
if Tmax <= 10
    thres = round((Totsu+Tmax)/4);
else
    thres = round((Totsu+Tmax)/2);
end
% Binarize
mask_hist = uint8(zeros(size(img(:, :, 1))));
if Tmax <= 200
    mask_hist(img_gray > thres) = 1;
else
    mask_hist(img_gray < thres) = 1;
end
% Skin Segmentation
mask_skin = skinSeg(img,mode);
mask = mask_skin .* mask_hist;
% Make Dataset
dataset = makeDataset(img,mask);
% Train K-means
[idx,C] =
kmeans(dataset,3,'MaxIter',100,'Distance','cityblock','Replicates',5,'D
isplay','final');

% figure; imshow(logical(mask_hist));
% figure; imshow(mask);
end
```

3. testSkin.m

```
function [idx] = testSkin(img,C,mode)
% Input
% img: RGB Image
% C: Centroid of Clusters
% mode: 'Proposed'

% Output
% idx: Index of Clustered Image

% RGB to Gray Scale
img_gray = rgb2gray(img);
% Get Otsu Threshold
Totsu = graythresh(img_gray)*255;
% Histogram
[n,~] = histcounts(img_gray,256);
[~,Tmax] = max(n);
% Calculate Threshold
if Tmax <= 10
    thres = round((Totsu+Tmax)/4);
else
    thres = round((Totsu+Tmax)/2);
end
% Binarize
mask_otsu = uint8(zeros(size(img(:, :, 1))));
```

```
if Tmax <= 200
    mask_otsu(img_gray > thres) = 1;
else
    mask_otsu(img_gray < thres) = 1;
end
% Skin Segmentation
mask_skin = skinSeg(img,mode);
mask = mask_skin .* mask_otsu;
% Make Dataset
dataset = makeDataset(img,mask);
% Test K-means
[~,idx] = pdist2(C,dataset,'euclidean','Smallest',1);
end
```

4. makeDataset.m

```
function dataset = makeDataset(img,mask)
img = double(img);
imgHSV = rgb2hsv(img);
imgYCbCr = rgb2ycbcr(img);
Hue = imgHSV(:,:,1)';
Cb = double(imgYCbCr(:,:,2));
Cr = double(imgYCbCr(:,:,3));

h = Hue(:);
cr = Cr(:)/240;
cb = Cb(:)/240;
m = mask';
m = double(m(:))/255;

dataset = [cr,cb,h,m];
end
```

5. cluster2mask.m

```
function mask = cluster2mask(idx,n,m,skin,bg,fg)

mask = reshape(idx,m,n)';
mask(mask==skin)=255;
mask(mask==bg)=0;
mask(mask==fg)=60;
mask = uint8(mask);
end
```

6. skinSeg.m

```
function [res] = skinSeg(img,mode)
[n,m,~] = size(img);
dimg = double(img);
R = dimg(:,:,1);
G = dimg(:,:,2);
B = dimg(:,:,3);

if mode == "NormRGB" || mode == "Proposed"
    sumRGB = R+G+B;
    normR = R ./ sumRGB;
    normG = G ./ sumRGB;

    mask4 = zeros(n,m); mask4((normR./normG)>1.185) = 1;
    if mode == "NormRGB"
        mask5 = zeros(n,m); mask5((R.*G)./(sumRGB.*sumRGB)>0.112) = 1;
```

```
        mask6 = zeros(n,m); mask6((R.*B)./(sumRGB.*sumRGB)>0.107) = 1;
    end
end

if mode == "HS-CbCr" || mode == "HS" || mode == "Proposed"
    HSV = rgb2hsv(img);
    H = HSV(:,:,1);
    S = HSV(:,:,2);

    mask12 = zeros(n,m); mask12(S>=0.0754 & S<=0.6093)=1;
    mask13 = zeros(n,m); mask13(H<=0.1 & 0.01<=H) = 1;
end

if mode == "CbCr" || mode == "Proposed" || mode == "HS-CbCr"
    YCbCr = rgb2ycbcr(img);
    Cb = double(YCbCr(:,:,2));
    Cr = double(YCbCr(:,:,3));
    mask17 = zeros(n,m); mask17(Cb>77&Cb<127) = 1;
    mask18 = zeros(n,m); mask18(Cr>133&Cr<173) = 1;
end

if mode == "CbCr"
    mask = mask17 & mask18;
elseif mode == "NormRGB"
    mask = mask4 & mask5 & mask6;
elseif mode == "HS"
    mask = mask12 & mask13;
elseif mode == "HS-CbCr"
    mask = mask12 & mask13 & mask17 & mask18;
elseif mode == "Proposed"
    mask = mask4 & mask12 & mask13 & mask17 & mask18 ;
end

res = 255 * uint8(mask);
end
```

7. motionDetection.m

```
function [bw] = motionDetection(img,prev_img)
% Frame Difference
Fdt = abs(img - prev_img);
% rgb2gray
R = Fdt(:, :, 1);
G = Fdt(:, :, 2);
B = Fdt(:, :, 3);
Fdg = 0.299*R + 0.587*G + 0.114*B;
% calculate T
X = mean(Fdg, 'all');
T = 0.05 * X;
Q = imbinarize(Fdg,T); % D(t,t+1)
% Morphological Image Processing
se = strel('square',5);

bw = imopen(Q,se);
bw = imclose(bw,se);
bw = imfill(bw,'holes');
bw = imclearborder(bw,4);
bw = bwareaopen(bw,300);
end
```