FINGER CNC

开放设计 客制未来

视觉功能
使用手册

版本号：F202409VU-CN

广州亿达科技有限公司

目 录

1. 文档概要说明

1.1 修改记录

修改时间	备注
20240507	初次修订手册

1.2 关于亿达

广州亿达科技有限公司旨在打造性能卓越的开放式数控系统，让自动化开发变得更简单、机床自动化触手可及。 作为中国高性能控制器制造商之一，亿达科技专注于客户需求，不断突破技术研发边界，逐步形成完善的自动化关键技术的生态系统，为客户提供全面解决方案和便捷服务。在中国多个地区，均建立了完整、专业、高效的销售和服务渠道。

亿达科技持续专注于数控系统、运动控制器、边缘计算控制器、Open CNC 开发平台、CAD/CAM 技术、机器视觉技术、机械手控制技术、工业物联网技术的研发和生产。领先行业的 Open CNC 开发平台，让机器设备客制化开发变得更简单，并为客户创造专属的产品价值。亿达科技提出建立 6 个核心技术内嵌（运动控制、HMI、PLC、机器视觉、CAD/CAM、物联网）作为我们产品的一体化解决方案，为客户提供最佳自动化解决方案。此外我们已经在车铣加工中心、磨床、弹簧机、刀具机、木工机械、绕线机、旋压机、弯管机、3C 电子等行业，积累了丰富的产品经验和客户基础，并不断创造卓越。

在高速高精领域，亿达科技深入研究多种高性能运动控制算法，广泛适用于不同行业需求，尤其是在多轴联动插补、RTCP 五轴联动控制、多轴多通道控制、电子凸轮、卷绕、张力等技术领域，为客户提供了更多可选择的解决方案。

专注需求、关注结果、卓越创新、尊重人才是亿达科技创立以来所秉持的经营理念和企业价值。一直以来，我们坚持初心，砥砺前行，不断研发好用、可靠的自动化产品，并将 Open CNC

理念做到客户终端，成就客户专属价值。

1.3 前言

在B系列控制器及以上的机型配备了嵌入式视觉的功能，并且，嵌入式视觉有C++开放式接口和Python脚本两种使用方法，本文主要介绍如何使用C++开放式接口来搭建符合生产需求的嵌入式视觉功能。

该文档可以作为未使用过视觉功能的用户的入门培训文档，也可以作为使用的过程中，对接口或者功能有疑问的查询手册。文档中的每一个接口都带有使用案例，每个案例都可以复制代码到HMI软件直接运行查看效果，用户对接口功能不清楚的，可以在使用案例的基础上，多尝试去设置不同的检测对象、不同的检测参数，对比不同的效果。

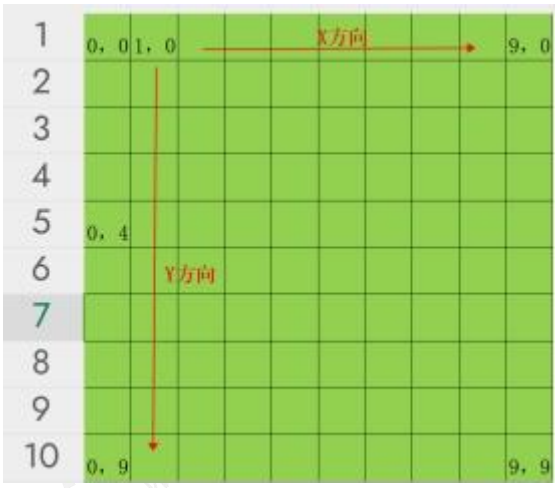
1.4 使用前需要确认的信息

嵌入式视觉功能目前主要通过 HMI 软件在开发画面工程时，由用户调用视觉的相关接口，根据实际生产需求，来完成视觉功能的开发。为了确保你的开发和使用正常，以下是开发前需要确认的信息：

1. 确认使用的控制器为 B 系列控制器及以上；
2. 确认 HMI 软件中，宏向导有相机模块 HCamera 和视觉处理模块 HVisionModule；
3. 确认控制器中的视觉功能已经开启。视觉功能需要通过开轴软件开启，可以通过检查变数 com40783，为 1 时开启，为 0 时未开启，不能使用视觉功能。
4. 确认你使用的相机是网口相机还是USB相机。如果是网口相机，请确保网线直连相机和控制器网口（通过交换机连接也可以），使用USB转网口有可能会导致连接错误。如果是USB相机，请确保相机的USB端连接的是控制器的USB3.0 的接口，而非USB2.0，否则连接有可能出现不稳定的现象。
5. 在使用网口相机时，请查看相机的IP和控制器的IP是否在同一个网段，否则无法连接。可以使用接口HCamera.getIpList()和HCamera.setIp(n, ip)获取和设置相机ip。

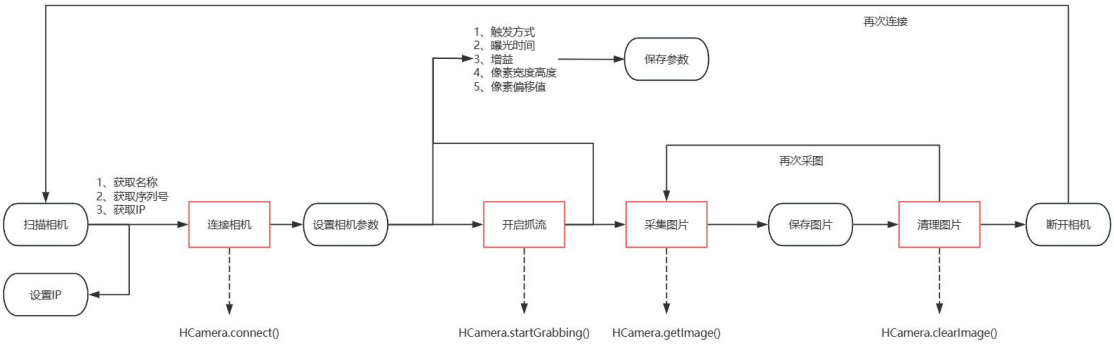
1.5 视觉基础概念

1. 相机分为彩色相机和黑白相机，彩色相机采集的图片具有三个颜色通道（R、G、B），黑白相机采集的图片只有一个颜色通道（Gray），因此黑白相机采集的灰度图片没有彩色。
2. 比较常用的是黑白相机，采集到的灰度图片可以理解为W*H的二维阵列（或二维矩阵），比如 1200W像素的黑白相机，分辨率为 4000（W）*3000（H），由相机的硬件内部决定。（4000*3000 = 1200 万，因此称为 1200W相机）
3. 在视觉领域，图像中左上角为原点、顶点，坐标为（0，0）。横向向右为X正方向，竖向向下为Y正方向。如下图（分辨率为 10*10）：



4. 在灰度图片中，每个像素存放一个“灰度级”，也就是 0-255 之间的整数。当灰度级为 0 时，图片在该像素显示黑色，当灰度级为 255 时，图片在该像素显示白色。0 到 255 之间，则是从黑到白的不同程度上的灰色。
5. 在 HCamera 和 HVisionModule 模块中，提及“图片”的都是一个变量，是一个整体。无论是读取“图片”、还是采集“图片”，还是接口中返回的类型是“图片”，都会根据图片本身所占的内存大小而上涨内存，因此在调用接口时需要留意内存的管理（使用对应模块的 clearImage 函数）。
6. 在使用 HCamera 模块前，请理解下图的流程和功能：

注意：正常的图片采集流程如下图：图中红色矩形为必须调用的接口，黑色圆角矩形为非必要调用的接口。在采图时，请确认触发模式。



2. 相机模块介绍

2.1 相机功能预览（ctrl+单击跳转）

- 获取相机名称
- 获取相机序列号
- 获取网口相机 IP 地址
- 设置网口相机 IP 地址
- 连接相机
- 断开相机
- 获取相机连接状态
- 获取相机曝光时间
- 设置相机曝光时间
- 获取相机增益
- 设置相机增益
- 获取相机触发方式
- 设置相机触发方式
- 采集图片
- 保存图片
- 获取帧率使能状态

- 设置帧率使能状态
- 获取帧率值
- 设置帧率值
- 开始抓流
- 停止抓流
- 获取像素宽度
- 设置像素宽度
- 获取像素高度
- 设置像素高度
- 获取像素宽度最大值
- 获取像素高度最大值
- 获取像素偏移值 X
- 获取像素偏移值 Y
- 设置像素偏移值 X
- 设置像素偏移值 Y
- 保存相机参数
- 清理图片所占内存
- 实时显示（视频流）
- 设置定时器方法

2.2 相机属性介绍

相机属性介绍	说明
曝光时间	是指从快门打开到关闭的时间间隔，曝光时间越长，图片亮度越高，可以根据图片的亮暗程度调整曝光时间的长短。
相机增益	相机增益是指在拍摄过程中增加传感器信号的强度，以提高图像亮度的过程。 然而，增加增益也会引入图像噪点，影响图像质量。因此，在选择增益时需要权衡图像亮度和噪点水平，以获得最佳的拍摄效果。

触发模式	触发模式主要有以下几种： 1. 连续触发：是指相机以设定的频率不断地捕捉图像数据，达到实时显示效果。 2. 软触发：是指通过相机的软件命令或者信号来触发相机捕捉图像。 3. 硬触发：通过外部的触发信号或者触发器来触发相机捕捉图像。
相机帧率	是指相机在 1 秒内拍摄下多少幅连续的画面,它的单位是fps,即 frame per second (每秒传输帧数)。
开始/停止抓流	开始抓流时，相机处于采图的预备状态，等待采图触发命令。在修改一些属性前，如设置像素宽度、高度、偏移值，需要停止抓流。在采图前，需要开始抓流。
像素宽度	相机采集图片的分辨率宽度，如 4000*3000 分辨率的图片的像素宽度为 4000。
像素高度	相机采集图片的分辨率高度，如 4000*3000 分辨率的图片的像素高度为 3000。
像素偏移值X	当图片的像素宽度被裁剪时，可以设置像素偏移值X控制视野的横向偏移
像素偏移值Y	当图片的像素高度被裁剪时，可以设置像素偏移值Y控制视野的纵向偏移

2.3 相机接口功能详情、使用案例、使用效果

2.3.1 获取相机名称

【接口】

```
var list = HCamera.getDeviceList()
```

【描述】

获取相机列表中的名称型号（当有相机处于连接状态时，无法更新扫描列表）

【参数】

无

【返回值】

(array)返回设备上的型号名称列表

【使用案例】

```
var list  = HCamera.getDeviceList()
var number = list.length
print("相机数量为：", number)
for(i = 0; i < number; i++)
{
    str1 = "相机" + i + ":"
    print(str1 + list[i])
}
```

【执行效果】

```
相机数量为：1
相机 0：MV-CS050-10UM
```

2.3.2 获取相机序列号

【接口】

```
var list = HCamera.getSnList()
```

【描述】

获取相机列表中的序列号。（当有相机处于连接状态时，无法更新扫描列表）

【参数】

无

【返回值】

(array)返回设备上的序列号列表

【使用案例】

```
var list  = HCamera.getSnList()
```

```
var number = list.length
print("相机数量为：", number)
for(i = 0; i < number; i++)
{
    str1 = "相机" + i + ":"
    print(str1 + list[i])
}
```

【执行效果】

扫描失败返回：

```
相机数量为：0
```

扫描成功返回：

```
相机数量为：1
相机 0：U3V:DA0888839
```

注意：此处以及后面所有的’print’语句都属于是后台打印（实际上不开放该打印），仅供参考。

2.3.3 获取网口相机 IP 地址

【接口】

```
var list = HCamera.getIpList()
```

【描述】

获取相机列表中的 IP。（当有相机处于连接状态时，无法更新扫描列表）

【参数】

无

【返回值】

(array)返回设备列表的 IP。仅网口相机有效。

【使用案例】

```
var list  = HCamera.getIpList()
var number = list.length
```

```
print("相机数量为：", number)
for(i = 0; i < number; i++)
{
    str1 = "相机" + i + ":"
    print(str1 + list[i])
}
```

【执行效果】

```
相机数量为：1
相机 0：192.168.2.97
```

2.3.4 设置网口相机 IP 地址

【接口】

```
var flag = HCamera.setIp(n, ip)
```

【描述】

设置相机的IP，在相机连接后无法使用。仅对网口相机有效。

【参数】

(int)n: 第几个设备。（从 0 开始）

(string)n: ip地址，如ip = "192.168.3.1"。

【返回值】

(bool)成功返回true，失败返回false

【使用案例】

```
var list  = HCamera.getIpList()
var number = list.length
print("相机数量为：", number)
for(i = 0; i < number; i++)
{
    str1 = "相机" + i + ":"
```

```
    print(str1 + list[i])
}
var flag = HCamera.setIp(0, "192.168.3.1")
print("设置ip是否成功flag = ", flag)
```

【执行效果】

```
相机数量为：1
相机 0：192.168.2.97
设置 ip 是否成功 flag = true
```

注意：设置相机IP可以设置不同网段，但是连接相机需要控制器与相机在同一网段。

2.3.5 连接相机

【接口】

```
var flag = HCamera.connect(n)
```

【描述】

开始连接相机。

【参数】

(int)n: 第几个设备。（从 0 开始）

【返回值】

(bool)成功返回true，失败返回false。

【使用案例】

```
var flag = HCamera.connect(0)
print("连接相机 flag = ", flag)
```

【执行效果】

```
连接相机 flag = true
```

注意：连接网口相机时，需要确保网口相机的IP与控制器IP处于同一个网段。

2.3.6 断开相机

【接口】

var flag = HCamera.disconnect(n)

【描述】

断开相机。

【参数】

(int)n：第几个设备。（从 0 开始）

【返回值】

(bool)成功返回true，失败返回false。

【使用案例】

```
var flag = HCamera.connect(0)
print("连接相机 flag = ", flag)
flag = HCamera.disconnect(0)
print("断开相机 flag = ", flag)
```

【执行效果】

```
连接相机 flag = true
断开相机 flag = true
```

2.3.7 获取相机连接状态

【接口】

```
var flag = HCamera.getCameraState(n)
```

【描述】

获取相机连接状态。

【参数】

(int)n: 第几个设备。

【返回值】

(bool)在线返回true，离线返回false。

【使用案例】

```
var flag = HCamera.getCameraState(0)
print("获取相机状态 flag = ", flag)

flag = HCamera.connect(0)
print("连接相机 flag = ", flag)

flag = HCamera.getCameraState(0)
print("获取相机状态 flag = ", flag)

flag = HCamera.disconnect(0)
print("断开相机 flag = ", flag)

flag = HCamera.getCameraState(0)
print("获取相机状态 flag = ", flag)
```

【执行效果】

```
获取相机状态 flag = false
连接相机 flag = true
获取相机状态 flag =true
断开相机 flag = true
获取相机状态 flag = false
```

2.3.8 获取相机曝光时间

【接口】

var time = HCamera.getExposureTime(n)

【描述】

获取相机的曝光时间，单位us

【参数】

(int)n: 第几个设备。（从 0 开始）

【返回值】

(double)返回相机的曝光时间（单位us），-1 为获取失败。

【使用案例】

```
var flag = HCamera.connect(0)
print("连接相机 flag = ", flag)
var time = HCamera.getExposureTime(0)
print("相机曝光时间为：", time)
```

【执行效果】

```
连接相机 flag = true
相机曝光时间为：9000
```

2.3.9 设置相机曝光时间

【接口】

var flag= HCamera.setExposureTime(n, dExposureTime)

【描述】

设置相机的曝光时间（单位us）。

【参数】

(int)n: 第几个设备。（从 0 开始）

(double)dExposureTime: 曝光时间（单位us）

【返回值】

(bool)成功返回true，失败返回false。

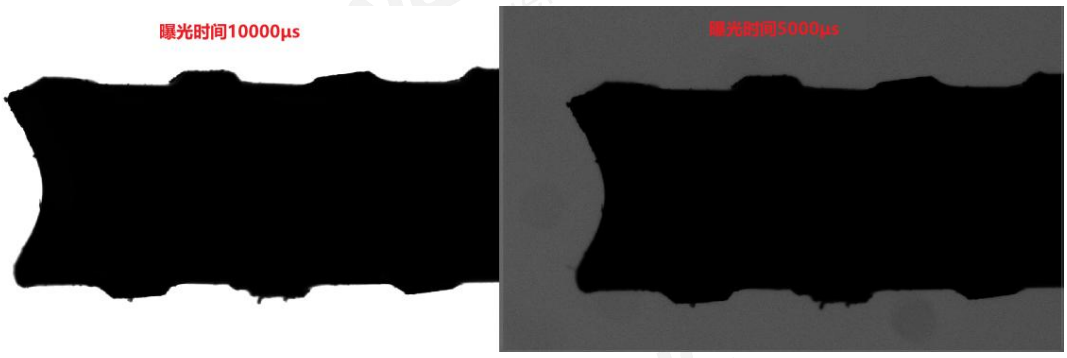
【使用案例】

```
var flag = HCamera.connect(0)
print("连接相机 flag = ", flag)
var flag2 = HCamera.setExposureTime(0, 5000)
print("修改曝光时间flag2 = ", flag2)
```

【执行效果】

```
连接相机 flag = true
修改曝光时间 flag2 = true
```

注意：曝光时间越大，图像的亮度越高，效果如下图



同时，加大曝光时间后，采集的帧率会下降，采集速度有所下降，所以一般不能设置太大。

并且，在拍摄运动的物体时，曝光时间越长，拍摄的图像越容易模糊有拖影，如下图，

补充专业知识：

假设物体的速度是 0.5m/s，相机是 640*480 的分辨率，视场为 4*3mm ；

3mm/480=0.006mm,因此像素当量为 0.006mm,

当物体在快门时间内的运动大于 1.5 个像素时我们就可以认为会出现拖影。

因此要不出现拖影则：t(曝光时间)=0.006*1.5/0.5=0.018s = 18ms

2.3.10 获取相机增益

【接口】

```
var gain = HCamera.getGain(n)
```

【描述】

获取相机的增益。

【参数】

(int)n: 第几个设备。（从 0 开始）

【返回值】

(double)相机的增益，-1 为获取失败。

【使用案例】

```
var flag = HCamera.connect(0)
print("连接相机 flag = ", flag)
var gain = HCamera.getGain(0)
print("相机增益值为： ", gain)
```

【执行效果】

```
连接相机 flag = true
相机增益值为： 2
```

2.3.11 设置相机增益

【接口】

```
var flag = HCamera.setGain(n, dGain)
```

【描述】

设置相机的增益。

【参数】

(int)n: 第几个设备。（从 0 开始）

(double)dGain: 相机的增益

【返回值】

(bool)成功返回true，失败返回false。

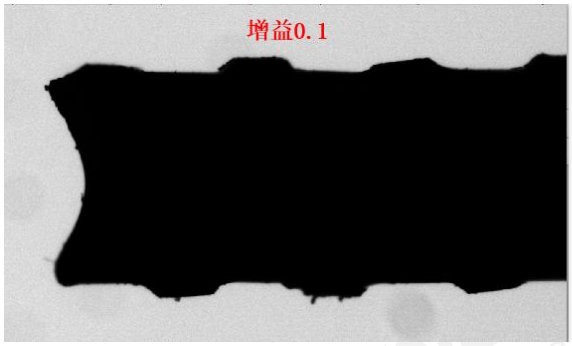
【使用案例】

```
var flag = HCamera.connect(0)
print("连接相机 flag = ", flag)
var flag2 = HCamera.setGain(0, 1)
print("修改增益 flag2 = ", flag2)
```

【执行效果】

```
连接相机 flag = true
修改增益 flag2 = true
```

增益相当于相机内部的一个图片效果放大器，可以成倍增加图片效果，其作用是在低光环境下成倍地提高图像亮度，但是增加增益也会使图像的噪点被放大，因此一般很少设置增益，都使用默认值 1。



2.3.12 获取相机触发方式

【接口】

var trig = HCamera.getTrigMode(n)

【描述】

获取相机的触发方式。

【参数】

(int)n: 第几个设备。（从 0 开始）

【返回值】

(int)返回相机的触发方式。=-1 获取失败，=0 连续触发， =1 软触发， =2 硬触发

【使用案例】

```
var flag = HCamera.connect(0)
print("连接相机 flag = ", flag)
var trig = HCamera.getTrigMode(0)
print("相机触发方式为： ", trig )
```

【执行效果】

```
连接相机 flag = true
相机触发方式为：1
```

2.3.13 设置相机触发方式

【接口】

var flag = HCamera.setTrigMode (n, trig)

【描述】

设置相机的触发方式。0（连续触发）、1（软触发）、2（硬触发）

【参数】

(int)n: 第几个设备。（从 0 开始）

(int)trigType: 相机的触发方式，0（连续触发）、1（软触发）、2（硬触发）

功能：设置相机的触发方式

【返回值】

(bool)成功返回true，失败返回false。

【使用案例】

```
var flag = HCamera.connect(0)
print("连接相机 flag = ", flag)
var flag2 = HCamera.setTrigMode (0, 1)
print("修改触发方式flag2 = ", flag2)
```

【执行效果】

```
连接相机 flag = true
修改触发方式 flag2 = true
```

注意：

1. 通过接口可以设置三种触发方式：连续触发、软触发（也叫单次触发）、硬件触发。采图与触发模式息息相关，采集图片的接口仅仅只是进行“取相”，而完整的拍照流程包括“触发命令”+“曝光、传输”+“取相”，因此，选择正确的触发方式对项目流程很重要。
2. 连续触发时，相机自主进行连续的、以帧率为触发频率，一直进行“触发命令 + 曝光 + 传输”的动作，CPU占用资源比较高。优点是采集速度快。
3. 软件触发时，相机等待采集图片接口HCamera.getImage (n)的触发命令，然后再进行一次“触发命令+曝光+传输+取相”的动作。因此CPU的占用资源较低，缺点是耗时会比连续触发多。
4. 硬件触发时，需要通过外接IO线给触发信号，IO触发一次，相机进行一次的“触发命令+曝光+传输”，然后通过采集图片接口HCamera.getImage (n)完成“取相”。

2.3.14 采集图片

【接口】

```
var image = HCamera.getImage (n)
```

【描述】

单帧采集，触发相机采集一张图片（需要相机的触发模式为 0 或 1）

【参数】

(int)n: 第几个设备。（从 0 开始）

【返回值】

(array)QVariant装载采集到的图片，失败返回NULL

【使用案例】

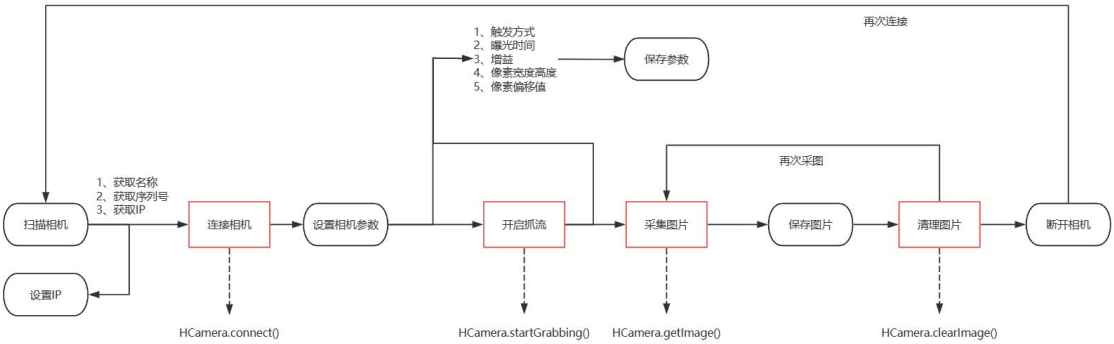
```
var flag = HCamera.connect(0)
print("连接相机 flag = ", flag)
flag = HCamera.startGrabbing(0)
print("开始抓流 flag = ", flag )
var image = HCamera.getImage (0)
var flag2 = HCamera.saveImage(image , "/home/image1.bmp")
print("保存图片 flag2 = ", flag2)
var flag3 = HCamera.clearImage(image)
print("清理图片内存 flag3 = ", flag3)
```

【执行效果】

```
连接相机 flag = true
开始抓流 flag = true
保存图片 flag2 = true
清理图片内存 flag3 = true
```

注意：

正常的图片采集流程如下图：图中红色矩形为必须调用的接口，黑色圆角矩形为非必要调用的接口。在采图时，请确认触发模式。



2.3.15 保存图片

【接口】

var image = HCamera.saveImage(img, path, imageFormat = null, quality = -1)

【描述】

保存一张getImage接口返回的图片，路径可自动创建。

【参数】

(array)img: getImage接口返回的图片，需要保存的图片变量

(string)path:图片保存路径，如Path = "/home/root/hust/usr/test/test.bmp"

(string)imageFormat="null":图片保存格式，如"BMP"、"JPG"、"PNG"等。有默认值null，且图片保存路径中包含了图片格式，一般不用填。

(int)quality=-1:图片保存质量，1（最低）~100（最高）。一般填-1 即可，代表 100。有默认值，一般不用填。

【返回值】

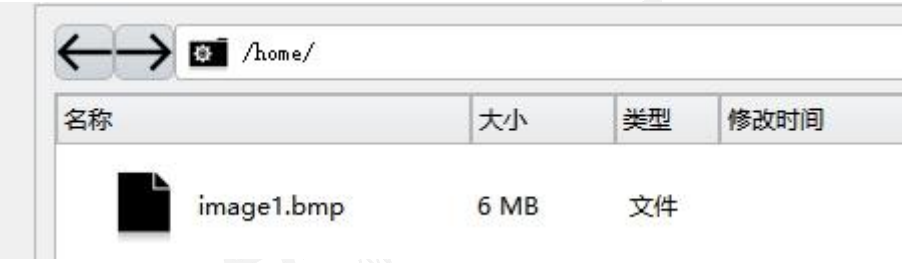
(bool)成功返回true，失败返回false。

【使用案例】

```
var flag = HCamera.connect(0)
print("连接相机 flag = ", flag)
flag = HCamera.startGrabbing(0)
print("开始抓流 flag = ", flag )
var image = HCamera.getImage (0)
var flag2 = HCamera.saveImage(image , "/home/image1.bmp")
print("保存图片 flag2 = ", flag2)
var flag3 = HCamera.clearImage(image)
print("清理图片内存 flag3 = ", flag3)
```

【执行效果】

```
连接相机 flag = true
开始抓流 flag = true
保存图片 flag2 = true
清理图片内存 flag3 = true
```



2.3.16 获取帧率使能状态

【接口】

```
var flag = HCamera.getFrameRateEnable(n)
```

【描述】

获取相机帧率设置使能。

【参数】

(int)n: 第几个设备。（从 0 开始）

【返回值】

(bool)使能状态

【使用案例】

```
var flag = HCamera.connect(0)
print("连接相机 flag = ", flag)
var flag2= HCamera.getFrameRateEnable(0)
print("相机帧率使能状态 flag2 = ", flag2)
```

【执行效果】

```
连接相机 flag = true
相机帧率使能状态 flag2 = true
```

2.3.17 设置帧率使能状态

【接口】

var flag = HCamera.setFrameRateEnable(n, enable)

【描述】

设置相机帧率使能状态。

【参数】

(int)n: 第几个设备。（从 0 开始）

(bool)enable:使能状态

【返回值】

(bool)成功返回true，失败返回false。

【使用案例】

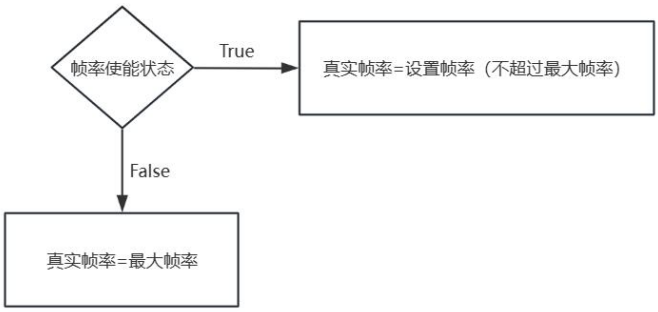
```
var flag = HCamera.connect(0)
print("连接相机 flag = ", flag)
var flag2 = HCamera.setFrameRateEnable(0, 1)
print("设置相机帧率使能状态是否成功 flag2 = ", flag2)
```

【执行效果】

```
连接相机 flag = true
设置相机帧率使能状态是否成功 flag2 = true
```

注意：

设置帧率使能状态的效果就是让用户设置的帧率是否生效，如果使能状态关闭，则默认相机以最大帧率采集传输。所谓帧率，指的是相机在 1 秒钟内拍摄下多少幅连续的画面，它的单位是fps，即 frame per second（每秒传输帧数）。



2.3.18 获取帧率值

【接口】

var value = HCamera.getFrameRate(n)

【描述】

获取相机帧率。

【参数】

(int)n: 第几个设备。（从 0 开始）

【返回值】

(double)相机当前帧率设置的帧率值

【使用案例】

```
var flag = HCamera.connect(0)
print("连接相机 flag = ", flag)
var value = HCamera.getFrameRate(0)
print("获取相机帧率 value = ", value)
```

【执行效果】

```
连接相机 flag = true
获取相机帧率 value = 60
```

2.3.19 设置帧率值

【接口】

var flag = HCamera.setFrameRate(n, value)

【描述】

设置相机帧率

【参数】

(int)n: 第几个设备。（从 0 开始）

(double)value：相机帧率

【返回值】

(bool)成功返回true，失败返回false。

【使用案例】

```
var flag = HCamera.connect(0)
print("连接相机 flag = ", flag)
var flag2 = HCamera.setFrameRate(0, 30)
print("设置相机帧率 flag2 = ", flag2)
```

【执行效果】

```
连接相机 flag = true
设置相机帧率 flag2 = true
```

注意：

帧率设置可以设置任何数值，但是实际帧率必然不会超过相机本身的最大帧率，一般的相机最大帧率为 100 以内。并且只有当帧率使能状态生效时，用户设置的帧率才生效，可以查看设置帧率使能状态的接口查看详情。

2.3.20 开始抓流

【接口】

```
var flag = HCamera.startGrabbing(n)
```

【描述】

开始抓流。相机需要开始抓流才能拍照。需要停止抓流才能保存参数

【参数】

(int)n: 第几个设备。（从 0 开始）

【返回值】

(bool)成功返回true，失败返回false

【使用案例】

```
var flag = HCamera.connect(0)
print("连接相机 flag = ", flag)
var flag2 = HCamera.startGrabbing(0)
print("开始抓流 flag2 = ", flag2)
```

【执行效果】

```
连接相机 flag = true
开始抓流 flag2 = true
```

注意：

在执行采集图片的接口前，必须先开始抓流。

2.3.21 停止抓流

【接口】

```
var flag = HCamera.stopGrabbing(n)
```

【描述】

停止抓流。相机需要开始抓流才能拍照。需要停止抓流才能保存参数

【参数】

(int)n: 第几个设备。（从 0 开始）

【返回值】

(bool)成功返回true，失败返回false

【示例】

```
flag = HCamera.stopGrabbing(0)
```

【使用案例】

```
var flag = HCamera.connect(0)
print("连接相机 flag = ", flag)
var flag2 = HCamera.startGrabbing(0)
print("开始抓流 flag2 = ", flag2)
var flag3 = HCamera.stopGrabbing(0)
print("停止抓流 flag3 = ", flag3)
```

【执行效果】

```
连接相机 flag = true
开始抓流 flag2 = true
停止抓流 flag3 = true
```

2.3.22 获取像素宽度

【接口】

var width = HCamera.getPixelWidth(n)

【描述】

获取相机像素宽度

【参数】

(int)n: 第几个设备。（从 0 开始）

【返回值】

(int)运行成功返回相机当前的像素宽度，失败返回 0

【使用案例】

```
var flag = HCamera.connect(0)
print("连接相机 flag = ", flag)
var width = HCamera.getPixelWidth(0)
print("相机像素宽度为 width = ", width )
```

【执行效果】

```
连接相机 flag = true
相机像素宽度为 width = 3072
```

2.3.23 设置像素宽度

【接口】

var flag = HCamera.setPixelWidth(n, value)

【描述】

设置相机像素宽度

【参数】

(int)n: 第几个设备。（从 0 开始）

(int)value：设置相机像素宽度，通过该接口裁剪图片的尺寸

【返回值】

(bool)成功返回true，失败返回false。

【使用案例】

```
var flag = HCamera.connect(0)
print("连接相机 flag = ", flag)
var flag2 = HCamera.setPixelWidth(0, 1000)
print("设置相机像素宽度 flag2  = ", flag2 )
```

【执行效果】

```
连接相机 flag = true
设置相机像素宽度 flag2  = true
```

假设相机的像素分辨率为 10*10，那么像素最大宽度为 10，可以使用该接口设置使能的像素宽度，如下图，把一个 10*10 的相机设置像素宽度、高度为 6*8，那么采集图片时，有效的区域为图中绿色区域，最终得到的图片的分辨率为 6*8。可以使用该接口改变相机的拍照视野、图片的尺寸，但不会影响到精度。

2.3.24 获取像素高度

【接口】

```
var height = HCamera.getPixelHeight(n)
```

【描述】

获取相机像素高度

【参数】

(int)n: 第几个设备。（从 0 开始）

【返回值】

(int)运行成功返回相机当前的像素高度，失败返回 0

【使用案例】

```
var flag = HCamera.connect(0)
print("连接相机 flag = ", flag)
var height = HCamera.getPixelHeight(0)
print("获取相机像素高度 height = ", height)
```

【执行效果】

```
连接相机 flag = true
获取相机像素高度 height = 2048
```

2.3.25 设置像素高度

【接口】

```
var flag = HCamera.setPixelHeight(n, value)
```

【描述】

设置相机像素高度

【参数】

(int)n: 第几个设备。（从 0 开始）

(int)value：设置相机像素高度，通过该接口裁剪图片的尺寸

【返回值】

(bool)成功返回true，失败返回false。

【使用案例】

```
var flag = HCamera.connect(0)
print("连接相机 flag = ", flag)
var flag2 = HCamera.setPixelHeight(0, 1000)
```

```
print("设置相机像素高度 flag2 = ", flag2)
```

【执行效果】

```
连接相机 flag = true
设置相机像素高度 flag2 = true
```

假设相机的像素分辨率为 10*10，那么像素最大高度为 10，可以使用该接口设置使能的像素高度，如下图，把一个 10*10 的相机设置像素宽度、高度为 6*8，那么采集图片时，有效的区域为图中绿色区域，最终得到的图片的分辨率为 6*8。可以使用该接口改变相机的拍照视野、图片的尺寸，但不会影响到精度。

2.3.26 获取像素宽度最大值

【接口】

```
var maxWidth = HCamera.getPixelMaxWidth(n)
```

【描述】

获取相机最大像素宽度

【参数】

(int)n: 第几个设备。

【返回值】

(int)运行成功返回相机最大像素宽度，失败返回 0

【使用案例】

```
var flag = HCamera.connect(0)
```

```
print("连接相机 flag = ", flag)
var maxWidth = HCamera.getPixelMaxWidth(0)
print("获取相机最大像素宽度 maxWidth = ", maxWidth )
```

【执行效果】

```
连接相机 flag = true
获取相机最大像素宽度 maxWidth = 3072
```

补充：每个相机的最大像素尺寸是固定的，由硬件确定的。

2.3.27 获取像素高度最大值

【接口】

```
var maxHeight = HCamera.getPixelMaxHeight(n)
```

【描述】

获取相机最大像素高度

【参数】

(int)n: 第几个设备。

【返回值】

(int)运行成功返回相机最大像素高度，失败返回 0

【使用案例】

```
var flag = HCamera.connect(0)
print("连接相机 flag = ", flag)
var maxHeight = HCamera.getPixelMaxHeight(0)
print("获取相机最大像素高度 maxHeight = ", maxHeight )
```

【执行效果】

```
连接相机 flag = true
获取相机最大像素高度 maxHeight = 2048
```

补充：每个相机的最大像素尺寸是固定的，由硬件确定的。

2.3.28 获取像素偏移值 X

【接口】

var offsetX = HCamera.getOffsetX(n)

【描述】

获取相机像素X方向的偏移

【参数】

(int)n: 第几个设备。

【返回值】

(int)运行成功返回相机当前像素X方向的偏移，失败返回-1

【使用案例】

```
var flag = HCamera.connect(0)
print("连接相机 flag = ", flag)
var offsetX = HCamera.getOffsetX(0)
print("获取相机像素X方向的偏移 offsetX = ", offsetX )
```

【执行效果】

```
连接相机 flag = true
获取相机像素 X 方向的偏移 offsetX = 0
```

2.3.29 获取像素偏移值 Y

【接口】

var offsetY = HCamera.getOffsetY(n)

【描述】

获取相机像素Y方向的偏移

【参数】

(int)n: 第几个设备。

【返回值】

(int)运行成功返回相机当前像素Y方向的偏移，失败返回-1

【使用案例】

```
var flag = HCamera.connect(0)
print("连接相机 flag = ", flag)
var offsetY = HCamera.getOffsetY(0)
print("获取相机像素Y方向的偏移 offsetY = ", offsetY )
```

【执行效果】

```
连接相机 flag = true
获取相机像素 X 方向的偏移 offsetY = 0
```

2.3.30 设置像素偏移值 X

【接口】

```
var flag = HCamera.setOffsetX(n, value)
```

【描述】

设置相机像素X方向的偏移。由于硬件的原因，某些偏移值不能被设置，会自动设置为最近值（向下取值）。如设置为 9，会自动设置成 8。设置完最好再获取一次查看当前值，如示例

【参数】

(int)n: 第几个设备。

(int)value：设置相机像素X方向的偏移值

【返回值】

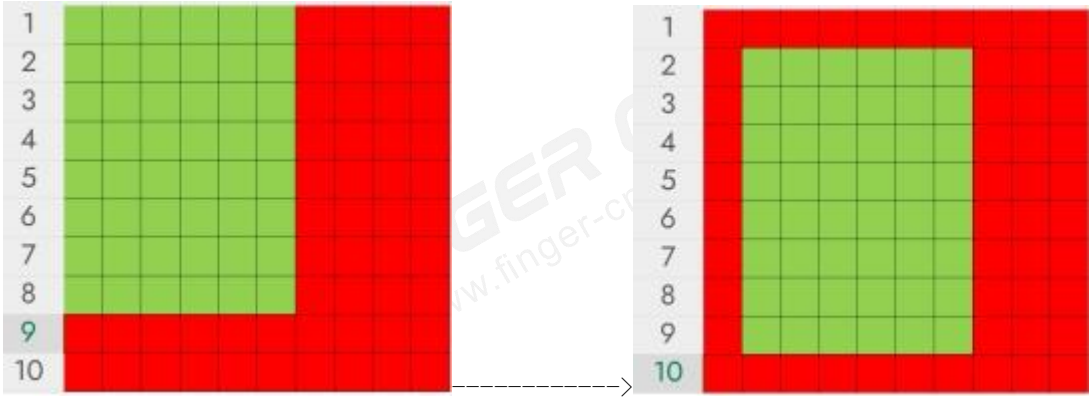
(bool)成功返回true，失败返回false。

【使用案例】

```
var offsetX = 30
flag = HCamera.setOffsetX(0, offsetX)
var currentValue = HCamera.getOffsetX(0)
```

【执行效果】

假设相机的像素分辨率为10*10，那么像素最大尺寸为10*10。假设使用设置像素宽高的接口修改为6*8，那么就可以设置偏移，如下图是设置X、Y偏移都为1的结果，生效的区域发生了偏移。X偏移值+像素宽度不能超过最大像素宽度。



2.3.31 设置像素偏移值 Y

【接口】

var flag = HCamera.setOffsetY(n, value)

【描述】

设置相机像素Y方向的偏移。由于硬件的原因，某些偏移值不能被设置，会自动设置为最近值（向下取值）。如设置为9，会自动设置成8。设置完最好再获取一次查看当前值，如示例

【参数】

(int)n: 第几个设备。

(int)value：设置相机像素Y方向的偏移值

【返回值】

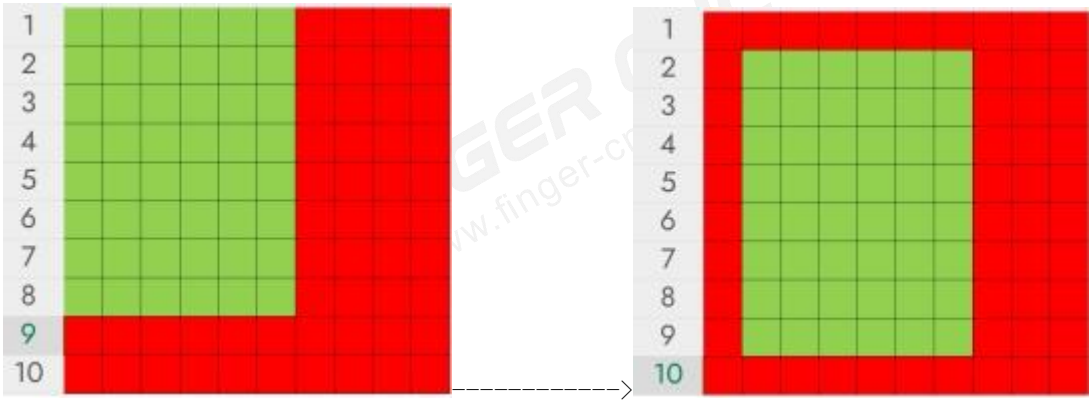
(bool)成功返回true，失败返回false。

【使用案例】

```
var offsetY = 30
flag = HCamera.setOffsetY(0, offsetY)
var currentValue = HCamera.getOffsetY(0)
```

【执行效果】

假设相机的像素分辨率为 10*10，那么像素最大尺寸为 10*10。假设使用设置像素宽高的接口修改为 6*8，那么就可以设置偏移，如下图是设置X、Y偏移都为 1 的结果，生效的区域发生了偏移。Y偏移值+像素高度不能超过最大像素高度。



2.3.32 保存相机参数

【接口】

```
var flag = HCamera.saveParameter(n)
```

【描述】

保存相机参数，相机断电后会自动读取该参数

【参数】

(int)n: 第几个设备。

【返回值】

(bool)成功返回true，失败返回false。

【使用案例】

```
var flag = HCamera.connect(0)
print("连接相机 flag = ", flag)
var flag2 = HCamera.setExposureTime(0, 5000)
print("修改曝光时间 flag2 = ", flag2)
var flag3 = HCamera.saveParameter(n)
```

```
print("保存参数标志 flag3 = ", flag3 )
```

【执行效果】

```
连接相机 flag = true
修改曝光时间 flag2 = true
保存参数标志 flag3 = true
```

2.3.33 清理图片所占内存

【接口】

```
var flag = HCamera.clearImage(image)
```

【描述】

手动清除图片内存。在相机的所有接口中，如果返回值为图片类型的，在使用完之后，都需要手动清除图片内存

【参数】

(array)image: 图片变量，通常由getImage()接口获取。

【返回值】

(bool)成功返回true，失败返回false。

【使用案例】

```
var flag = HCamera.connect(0)
print("连接相机 flag = ", flag)
flag = HCamera.startGrabbing(0)
print("开始抓流 flag = ", flag )
var image = HCamera.getImage (0)
var flag2 = HCamera.saveImage(image , "/home/image1.bmp")
print("保存图片flag2 = ", flag2)
var flag3 = HCamera.clearImage(image)
print("清理图片内存flag3 = ", flag3)
```

【执行效果】

```
连接相机 flag = true
开始抓流 flag = true
保存图片 flag2 = true
清理图片内存 flag3 = true
```

2.3.34 实时显示（视频流）

【描述】

实现实时显示需要定义两个宏（命名自定义），如下图所示，ccd_startLive通过与控件绑定作为开关使用，在全局变数定义一个标识符liveFlag=false（默认关闭），true为开启，false为关闭；ccd_cameraLive设置计时器，周期性获取图片并显示，达到实时显示的效果；

【使用案例】

(1) ccd_startLive实现：

```
var flag = HCamera.connect(0)
print("连接相机 flag = ", flag)
If (!liveFlag )
{
HCamera.startGrabbing(0)
    liveFlag = true                          //liveFlag 需在全局变数定义
} else
{
    liveFlag = false
    HCamera.stopGrabbing(0)
}
```

(2) ccd_cameraLive实现：

```
//Form_3.hVisionView 为当前显示界面插件的句柄
if(liveFlag )
```

```
{
    var image = HCamera.getImage(0)
    Form_3.hVisionView.showImagheByData(image)
    HVisionModule.clearImage(image)
}
```

2.3.35 设置定时器方法

1. 鼠标右键点击需要设置定时器的宏，点击设置。

2. 设置界面找到计时器栏，设置周期即可。

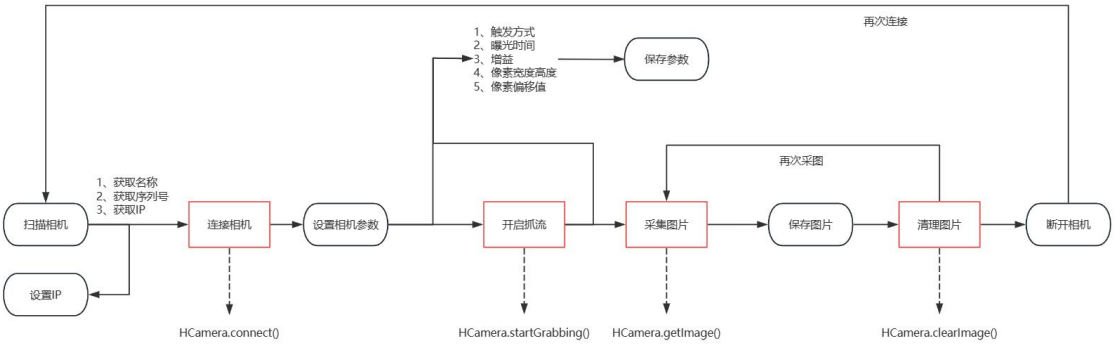
注：计时器周期越小，实时显示帧率越高，同时对CPU负荷越大，因此曝光时间也会受限。

计时器周期（ms）	建议曝光时间（us）
100	1000~50000
150	1000~50000

200	1000~100000
250	1000~120000
300	1000~140000

2.4 相机模块注意事项

注意理解以下的接口使用流程，采集图片时，红色步骤为必须调用的，黑色步骤为非必要调用的。



3. 图像处理模块介绍

3.1 图像处理模块功能预览（ctrl+单击跳转）

- 读取图片
- 相机模块图片变量转为视觉处理模块图片变量
- 颜色转换
- 获取图片最大轮廓点集
- 获取轮廓点集的正矩形
- 获取轮廓点集的最小外接矩形（面积最小）
- 绘制轮廓点集
- 绘制矩形

- 绘制矩形 2
- 绘制线段
- 绘制起点
- 绘制终点
- 绘制圆
- 绘制文本
- 裁剪图片
- 矩形转换为四个顶点坐标
- 筛选范围内的轮廓点集
- 合并轮廓点集
- 获取圆弧
- 设置绘制颜色
- 设置绘制线宽
- 阈值分割
- 开运算
- 图片相减
- 获取图片的所有物体的正矩形列表
- 获取图片平均哈希值
- 对比图片哈希值
- 保存图片
- 清理图片内存
- 设置绘制字体大小
- 边缘检测
- 获取拟合直线
- 拟合直线的相对坐标转换
- 获取直线距离
- 获取直线夹角
- 图片数据转换
- 获取白色区域面积

- 求线段与轮廓的切点及距离
- 模板匹配
- 图像缩放
- 图像旋转

3.2 图像处理模块接口功能详情、使用案例、使用效果

3.2.1 读取图片

【接口】

```
var image = HVisionModule.readImage(path, type)
```

【描述】

从本地读取一张图片

【参数】

(string)path: 图片路径，如path = "/home/hust/root/usr/image.bmp"

(int)type: 读取类型，type=0 代表以灰度类型读取，=1 代表以彩色类型读取

【返回值】

(array)运行成功返回image读取到的图片变量，运行失败返回空。

【示例】

```
var image = HVisionModule.readImage("/home/test.bmp", 0)
var flag = HVisionModule.clearImage(image)
print("清理图片内存 flag = ", flag)
```

【执行效果】

输入路径不正确时：

```
清理图片内存 flag = false
```

输入路径正确时：

```
清理图片内存 flag = true
```

注意：读取图片的接口，返回值是一个内存中的图片变量，因此在使用完之后，记得需要调用HVisionModule.clearImage(image) 接口进行清理，否则不断刷新堆积可能会导致内存泄漏问题。

3.2.2 相机模块图片变量转为视觉处理模块图片变量

【接口】

var visionImage = HVisionModule.cameraImageToVisionImage(cameraImage)

【描述】

视觉处理模块的接口不能直接使用相机模块的图片变量，需要用此接口进行转换。

【参数】

(var)cameraImage: 相机采集的图片变量，通常由HCamera.getImage()返回。

【返回值】

(array)运行成功返回HVisionModule可以处理的图片变量，运行失败返回空。

【使用案例】

```
var flag = HCamera.connect(0)
print("连接相机 flag = ", flag)
var flag2 = HCamera.startGrabbing(0)
print("开始抓流 flag2 = ", flag2)
var src = HCamera.getImage(0)
var image = HVisionModule.cameraImageToVisionImage(src)
flag = HCamera.clearImage(src )
print("清理图片内存flag = ", flag)
flag2 = HVisionModule.clearImage(image)
print("清理图片内存flag2 = ", flag2)
```

【执行效果】

```
连接相机 flag = true
开始抓流 flag2 = true
清理图片内存 flag = true
清理图片内存 flag2 = true
```

注意：如果是HCamera模块的接口返回的图片变量，则系统自动清理；如果是HVisionModule模块的接口返回的图片变量，需要使用HVisionModule模块的clearImage()接口清理；

3.2.3 颜色转换

【接口】

var image = HVisionModule.cvtColor(imageInput, type)

【描述】

转换图片类型

【参数】

(array)image: 输入图片变量

(int)type: 转换类型，type = 0 代表彩色转灰度，type = 1 代表灰度转彩色

【返回值】

(array)运行成功返回image转换后的图片，运行失败返回空。如果传入的图片类型不对，则返回原图。

【使用案例】

```
var image = HVisionModule.readImage("/home/test.bmp", 0)  //以灰度图片读入
var imageColor = HVisionModule.cvtColor(image , 1)        //转成彩色图片
var flag = HVisionModule.clearImage(image)
print("清理灰度图片内存flag = ", flag)
var flag2 = HVisionModule.clearImage(imageColor)
print("清理图片内存flag2 = ", flag2)
```

【执行效果】

```
清理灰度图片内存 flag = true
清理图片内存 flag2 = true
```

3.2.4 获取图片最大轮廓点集

【接口】

var contour = HVisionModule.getMaxContours(image, thr, type)

【描述】

获取图片中最大的轮廓点集合

【参数】

(array)image: 输入图片变量

(int)thr: 分割轮廓时的阈值，范围为 0-255

(int)type: 提取轮廓的类型，type=0 代表白色背景黑色物体，type=1 代表黑色背景白色物体

【返回值】

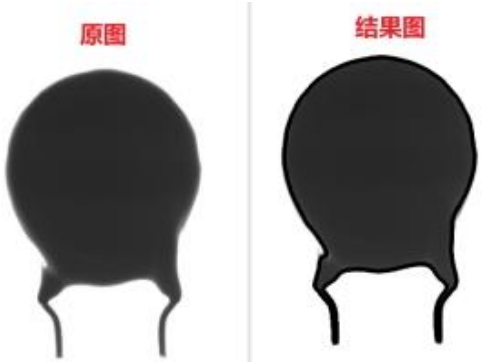
(array)运行成功返回contour获取到的图片中最大的轮廓，运行失败返回空

【使用案例】

```
var image = HVisionModule.readImage("/home/test.bmp", 0)
var contour = HVisionModule.getMaxContours(image, 100, 0)
var flag = HVisionModule.drawContours (image, contour, -1)
print("绘制flag = ",flag)
flag = HVisionModule.saveImage(image, "/home/contour.bmp")
print("保存flag = ",flag)
var flag2 = HVisionModule.clearImage(image)
```

【执行效果】

```
绘制 flag = true
保存 flag = true
```



3.2.5 获取轮廓点集的正矩形

【接口】

var list = HVisionModule.getBoundingRect(contour)

【描述】

获取轮廓点集合的外围正矩形位置，返回矩形的中心坐标(centerX, centerY)和长宽w、h和角度 0

【参数】

(array)contour: 输入轮廓点集合

【返回值】

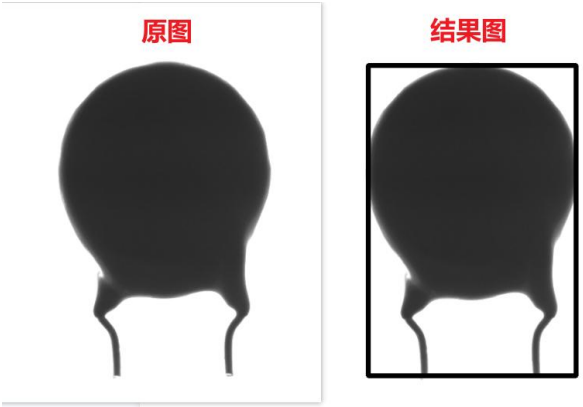
(array)运行成功返回rect包含了[centerX,centerY,w,h,0]的列表，运行失败返回空

【使用案例】

```
var image = HVisionModule.readImage("/home/test.bmp", 0)
var contour = HVisionModule.getMaxContours(image, 100, 0)
var list = HVisionModule.getBoundingRect(contour)
print(list)
var flag = HVisionModule.drawRectangle2(image, list)
flag = HVisionModule.saveImage(image, "/home/contour.bmp")
print("保存flag = ",flag)
flag = HVisionModule.clearImage(image)
```

【执行效果】

```
list:474,503,297,444,0
保存 flag = true
```



3.2.6 获取轮廓点集的最小外接矩形（面积最小）

【接口】

```
var list = HVisionModule.getMinAreaRect(contour)
```

【描述】

获取轮廓最小外接矩形的位置，输出的位置里包括中心坐标centerX，中心坐标cneterY,矩形长度w，矩形宽度h，旋转角度angle。

【参数】

(array)contour：输入轮廓点集合

【返回值】

(array)运行成功返回输出的位置结果列表[centerX,cneterY,w,h,angle],运行失败返回空

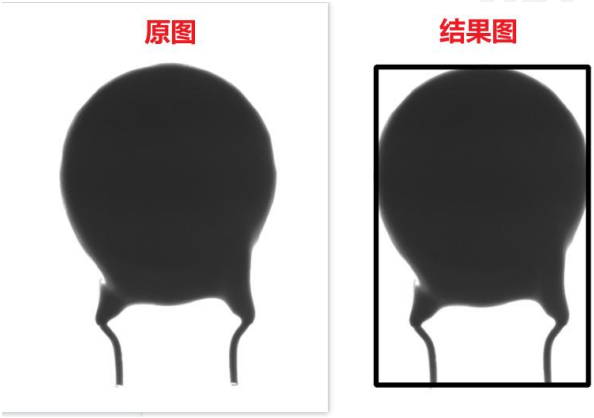
【使用案例】

```
var image = HVisionModule.readImage("/home/test.bmp", 0)
var contour = HVisionModule.getMaxContours(image, 100, 0)
var list = HVisionModule.getMinAreaRect (contour)
print(list)
var flag = HVisionModule.drawRectangle2(image, list)
flag = HVisionModule.saveImage(image, "/home/contour.bmp")
```

```
print("保存flag = ",flag)
flag = HVisionModule.clearImage(image)
print("清理flag = ",flag)
```

【执行效果】

```
list:474,503,297,444,0
保存 flag = true
```



3.2.7 绘制轮廓点集

【接口】

```
var flag = HVisionModule.drawContours(image, contour, num)
```

【描述】

在图片上绘制轮廓点集合

【参数】

(array)image: 输入图片，在该图片上绘制

(array)contour: 需要绘制的轮廓点集合

(int)num: 绘制轮廓的第几点，设置-1 时绘制全部轮廓点

【返回值】

(bool)成功返回true，失败返回false。

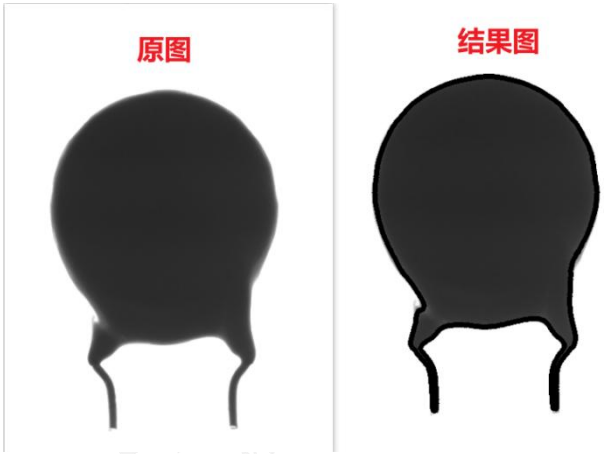
【使用案例】

```
var image = HVisionModule.readImage("/home/test.bmp", 0)
```

```
var contour = HVisionModule.getMaxContours(image, 100, 0)
var flag = HVisionModule.drawContours (image, contour, -1)
print("绘制flag = ",flag)
flag = HVisionModule.saveImage(image, "/home/contour.bmp")
print("保存flag = ",flag)
var flag2 = HVisionModule.clearImage(image)
```

【执行效果】

```
绘制 flag = true
保存 flag = true
```



3.2.8 绘制矩形

【接口】

```
var flag = HVisionModule.drawRectangle(image, pointX, pointY, w, h)
```

【描述】

在图片上绘制矩形

【参数】

(array)image: 输入图片，在该图片上绘制

(int)pointX: 绘制的起点坐标X

(int)pointY: 绘制的起点坐标Y

(int)w: 绘制的矩形的长度

(int)h: 绘制的矩形的高度

【返回值】

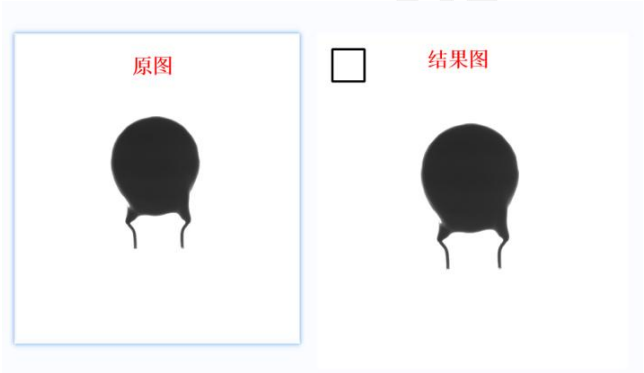
(bool)成功返回true，失败返回false。

【使用案例】

```
var image = HVisionModule.readImage("/home/test.bmp", 0)
var flag = HVisionModule.drawRectangle (image, 50, 50, 100, 100)
print("绘制flag = ",flag)
flag = HVisionModule.saveImage(image, "/home/rectImage.bmp")
print("保存flag = ",flag)
var flag2 = HVisionModule.clearImage(image)
```

【执行效果】

```
绘制 flag = true
保存 flag = true
```



3.2.9 绘制矩形 2

【接口】

```
var flag = HVisionModule.drawRectangle2(image, rectList)
```

【描述】

在图片上绘制矩形

【参数】

(array)image: 输入图片，在该图片上绘制

(array)rectList: 矩形列表,兼容正矩形与斜矩形,统一列表格式为[centerX, centerY, w, h,angle]，代表矩形的中心坐标X、Y、宽度、高度、角度

【返回值】

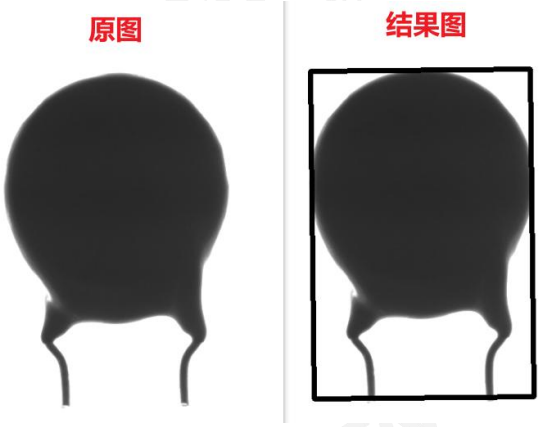
(bool)成功返回true，失败返回false。

【使用案例】

```
var image = HVisionModule.readImage("/home/test.bmp", 0)
var contour = HVisionModule.getMaxContours(image, 100, 0)
var list = HVisionModule.getMinAreaRect (contour)
var flag = HVisionModule.drawRectangle2(image, list)
flag = HVisionModule.saveImage(image, "/home/minRect.bmp")
print("保存flag = ",flag)
var flag2 = HVisionModule.clearImage(image)
```

【执行效果】

保存 flag = true



3.2.10 绘制线段

【接口】

```
var flag = HVisionModule.drawLine(image, x1, y1, x2, y2, lineType = 0)
```

【描述】

在图片上绘制线段

【参数】

(array)image: 输入图片，在该图片上绘制

(int)x1：线段的起点坐标x

(int)y1：线段的起点坐标y

(int)x2：线段的终点坐标x

(int)y2：线段的终点坐标y

(int)lineType：线的类型，默认为实线 0

【返回值】

(bool)成功返回true，失败返回false。

【使用案例】

```
var image = HVisionModule.readImage("/home/test.bmp", 0)
var flag = HVisionModule.drawLine (image, 50, 50, 100, 100)
flag = HVisionModule.saveImage(image, "/home/line.bmp")
print("保存flag = ",flag)
flag = HVisionModule.clearImage(image)
```

【执行效果】

保存 flag = true

3.2.11 绘制起点

【接口】

var flag = HVisionModule.moveTo(image, pointX, pointY)

【描述】

在图片上绘制路线，设置为起点，结合lineTo()接口使用可以绘制路线

【参数】

(array)image: 输入图片，在该图片上绘制

(int)pointX：起点的X坐标

(int)pointY：起点的Y坐标

【返回值】

(bool)成功返回true，失败返回false。

【使用案例】

```
var image = HVisionModule.readImage("/home/test.bmp", 0)
var flag = HVisionModule.moveTo(image, 50, 50)
flag = HVisionModule.lineTo(image, 100, 100)
flag = HVisionModule.saveImage(image, "/home/moToLineTo.bmp")
print("保存flag = ",flag)
flag = HVisionModule.clearImage(image)
```

【执行效果】

保存 flag = true

3.2.12 绘制终点

【接口】

var flag = HVisionModule.lineTo(image, pointX, pointY)

【描述】

在图片上绘制路线，设置为终点，以moveTo()接口的位置为起点，或以上一个lineTo()接口的位置为起点，绘制一条线段

【参数】

(array)image: 输入图片，在该图片上绘制

(int)pointX：终点的X坐标

(int)pointY：终点的Y坐标

【返回值】

(bool)成功返回true，失败返回false。

【使用案例】

```
var image = HVisionModule.readImage("/home/test.bmp", 0)
var flag = HVisionModule.moveTo(image, 50, 50)
flag = HVisionModule.lineTo(image, 100, 100)
flag = HVisionModule.saveImage(image, "/home/moToLineTo.bmp")
print("保存flag = ",flag)
flag = HVisionModule.clearImage(image)
```

【执行效果】

保存 flag = true

3.2.13 绘制圆

【接口】

```
var flag = HVisionModule.drawCircle(image, circleX, circleY, radius)
```

【描述】

在图片上绘制圆

【参数】

(array)image: 输入图片，在该图片上绘制

(int)circleX：圆心的坐标x

(int)circleY：圆心的坐标y

(int)radius：圆的半径

【返回值】

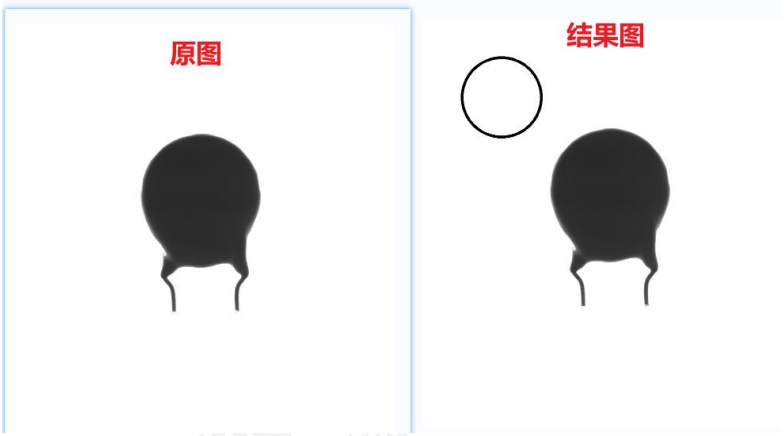
(bool)成功返回true，失败返回false。

【使用案例】

```
var image = HVisionModule.readImage("/home/test.bmp", 0)
var flag = HVisionModule.drawCircle(image, 200, 200, 100)
flag = HVisionModule.saveImage(image, "/home/cir.bmp")
print("保存flag = ",flag)
flag = HVisionModule.clearImage(image)
```

【执行效果】

```
保存 flag = true
```



3.2.14 绘制文本

【接口】

var flag = HVisionModule.drawText(image, text, pointX, pointY)

【描述】

在图片上绘制文字

【参数】

(array)image: 输入图片，在该图片上绘制

(string)text：需要绘制的文字

(int)pointX：显示的位置坐标x

(int)pointY：显示的位置坐标y

【返回值】

(bool)成功返回true，失败返回false。

【使用案例】

```
var image = HVisionModule.readImage("/home/test.bmp", 0)
var flag = HVisionModule.drawText(image, "OK", 100, 100)
flag = HVisionModule.saveImage(image, "/home/text.bmp")
print("保存flag = ",flag)
flag = HVisionModule.clearImage(image)
```

【执行效果】

保存 flag = true

3.2.15 裁剪图片

【接口】

var dst = HVisionModule.cutImage(imageInput, x, y, w, h)

【描述】

裁剪图片

【参数】

(array)imageInput：输入图片

(int)x：裁剪的起点坐标x

(int)y：裁剪的起点坐标y

(int)w：裁剪的长度w

(int)h：裁剪的高度h

【返回值】

(array)运行成功返回image裁剪完成后的图片，运行失败返回空

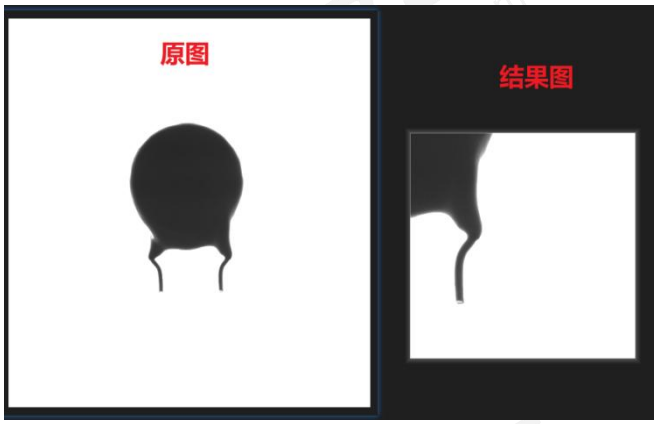
【使用案例】

```
var image = HVisionModule.readImage("/home/test.bmp", 0)
var dst = HVisionModule.cutImage(image, 50, 50, 100, 100)
var flag = HVisionModule.saveImage(dst, "/home/cut.bmp")
```

```
print("保存flag = ",flag)
flag = HVisionModule.clearImage(image)
flag = HVisionModule.clearImage(dst)
```

【执行效果】

保存 flag = true



3.2.16 矩形转换为四个顶点坐标

【接口】

```
var pointList = HVisionModule.rectToPoints(rectList)
```

【描述】

把矩形转换成四个顶点，返回到QVariantList中，按照以左上角为起点，逆时针的顺序返回

【参数】

(array)rectList：输入矩形的位置列表[centerX,centerY,,w,h,angle]

【返回值】

(array)运行成功返回point最小外接矩形的四个顶点的坐标[x1,y1,x2,y2,x3,y3,x4,y4]，运行失败返回空

【使用案例】

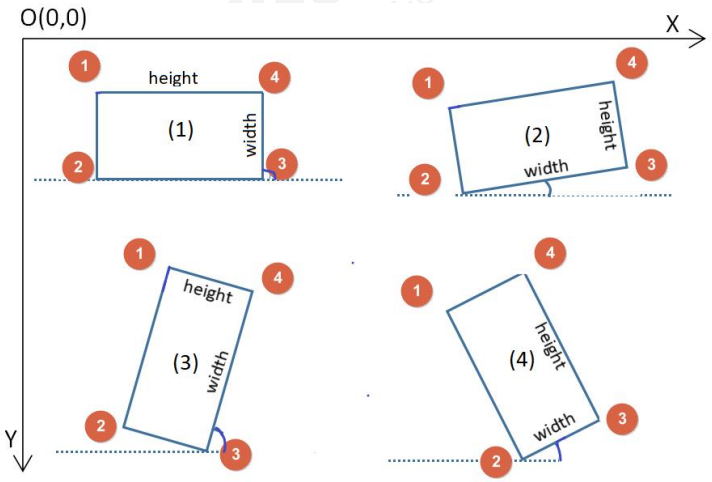
```
var image = HVisionModule.readImage("/home/test.bmp", 0)
var contour = HVisionModule.getMaxContours(image, 100, 0)
var list = HVisionModule.getMinAreaRect (contour)
```

```
var pointList = HVisionModule.rectToPoints(list )
print("顶点坐标为:", pointList)
flag = HVisionModule.clearImage(image)
```

【执行效果】

顶点坐标为：625,721,620,279,324,282,329,724

最小外接矩形的四个顶点坐标顺序如下图所示，以矩形左上角①为起始点，逆时针方向获取



3.2.17 筛选范围内的轮廓点集

【接口】

```
var newContour = HVisionModule.getRangeContour(contour, type, min, max)
```

【描述】

筛选轮廓点。输入一个轮廓点集合，可以通过type选择x方向或者y方向的最大最小值进行筛选轮廓点，得到在范围内的轮廓点集合。

【参数】

(array)contour：原轮廓点集合

(int)type：计算类型，0：对x方向进行筛选；1：对y方向进行筛选

(int)min：轮廓点的x坐标或y坐标的最小值

(int)max：轮廓点的x坐标或y坐标的最大值

【返回值】

(array)运行成功返回符合范围内的轮廓点集合，运行失败返回空

【使用案例】

```
var image = HVisionModule.readImage("/home/test.bmp", 0)
var contour = HVisionModule.getMaxContours(image, 100, 0)
var newContour = HVisionModule.getRangeContour(contour, 0, 100, 500)
var flag = HVisionModule.drawContours (image, newContour, -1)
print("绘制flag = ",flag)
flag = HVisionModule.saveImage(image, "/home/contour.bmp")
print("保存flag = ",flag)
flag = HVisionModule.clearImage(image)
```

【执行效果】

```
绘制 flag = true
保存 flag = true
```

3.2.18 合并轮廓点集

【接口】

```
var newContour = HVisionModule.mergeContour(contour1, contour12)
```

【描述】

输入两个轮廓点集合，合并成一个轮廓点集合

【参数】

(array)contour1：轮廓点集合 1

(array)contour2：轮廓点集合 2

【返回值】

(array)运行成功返回符合范围内的轮廓点集合，运行失败返回空

【使用案例】

```
var image = HVisionModule.readImage("/home/test.bmp", 0)
var contour = HVisionModule.getMaxContours(image, 100, 0)
```

```
var newContour = HVisionModule.getRangeContour(contour, 0, 100, 500)
var newContour2 = HVisionModule.getRangeContour(contour, 0, 1000, 1500)
var resultContour = HVisionModule.mergeContour(newContour, newContour2)
var flag = HVisionModule.drawContours (image, resultContour , -1)
print("绘制flag = ",flag)
flag = HVisionModule.saveImage(image, "/home/contour.bmp")
print("保存flag = ",flag)
flag = HVisionModule.clearImage(image)
```

【执行效果】

```
绘制 flag = true
保存 flag = true
```

3.2.19 获取圆弧

【接口】

```
var arcList = HVisionModule.getArc(contour, num)
```

【描述】

找圆弧功能模块

【参数】

(array)contour：输入轮廓点集合，使用轮廓中的点计算圆弧

(int)num：输入参数，计算迭代次数

【返回值】

(array)运行成功返回arc圆弧的圆心和半径列表[x,y,r]，运行失败返回空

【使用案例】

```
var image = HVisionModule.readImage("/home/test.bmp", 0)
var contour = HVisionModule.getMaxContours(image, 100, 0)
var arcList = HVisionModule.getArc(contour, 20)
print("圆弧圆心和半径为:", arcList )
```

```
flag = HVisionModule.clearImage(image)
```

【执行效果】

圆弧圆心和半径为：503,495,106

3.2.20 设置绘制颜色

【接口】

HVisionModule.setPenColor(image, r, g, b)

【描述】

设置图片绘制的颜色，注意必须输入彩色图片才有颜色

【参数】

(array)image：输入彩色图片，指定该图片当前绘制操作的颜色

(int)r：绘制的颜色r，红色通道数值，应该在 0-255 之间

(int)g：绘制的颜色g，绿色通道数值，应该在 0-255 之间

(int)b：绘制的颜色b，蓝色通道数值，应该在 0-255 之间

【返回值】

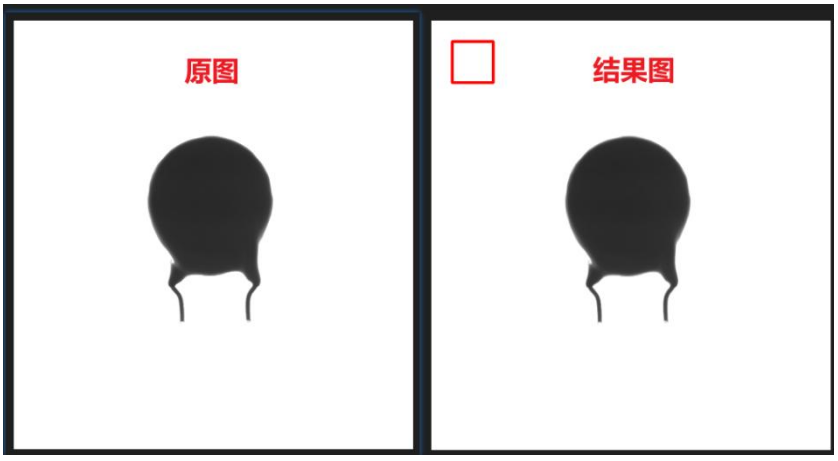
(bool)运行成功返回true，运行失败返回false

【使用案例】

```
var image = HVisionModule.readImage("/home/test.bmp", 0)
var imageColor = HVisionModule.cvtColor(image , 1)
HVisionModule.setPenColor(imageColor , 255, 0, 0)
var flag = HVisionModule.drawRectangle (imageColor , 50, 50, 100, 100)
flag = HVisionModule.saveImage(imageColor , "/home/result.bmp")
print("保存flag = ",flag)
flag = HVisionModule.clearImage(image)
flag = HVisionModule.clearImage(imageColor)
```

【执行效果】

保存 flag = true



3.2.21 设置绘制线宽

【接口】

var flag = HVisionModule.setThickness(image, thickness)

【描述】

设置所有绘制的线宽

【参数】

(array)image：输入图片，指定该图片当前绘制操作的线宽

(int)thickness：绘制的线宽，单位为像素

【返回值】

(bool)运行成功返回true，运行失败返回false

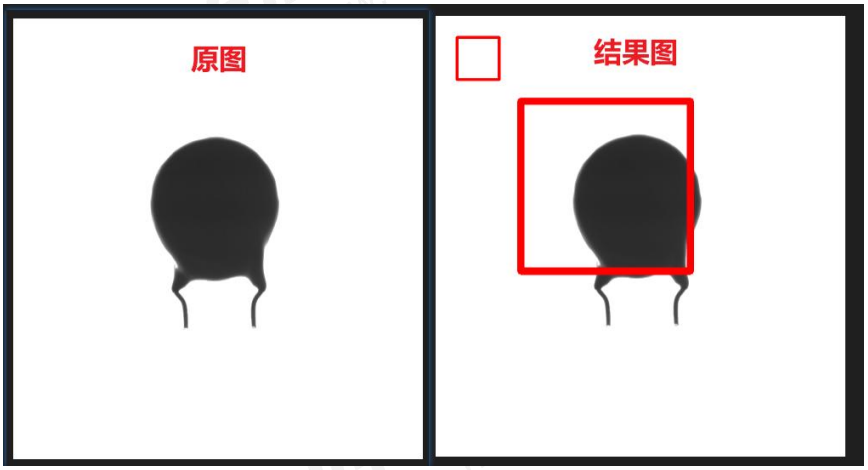
【使用案例】

```
var image = HVisionModule.readImage("/home/test.bmp", 0)
var imageColor = HVisionModule.cvtColor(image , 1)
HVisionModule.setPenColor(imageColor , 255, 0, 0)
var flag = HVisionModule.setThickness(imageColor , 5)
flag = HVisionModule.drawRectangle (imageColor , 50, 50, 100, 100)
var flag = HVisionModule.setThickness(imageColor , 15)
```

```
flag = HVisionModule.drawRectangle (imageColor , 200, 200, 400, 400)
flag = HVisionModule.saveImage(imageColor , "/home/result.bmp")
print("保存flag = ",flag)
flag = HVisionModule.clearImage(image)
flag = HVisionModule.clearImage(imageColor)
```

【执行效果】

保存 flag = true



3.2.22 阈值分割

【接口】

```
var thrImage = HVisionModule.thresholdProcessed(image, threshold, gray, type)
```

【描述】

阈值分割处理，把灰度图分割成二值图（只有 0 和 255）

【参数】

(array)image：输入灰度图片变量

(int)threshold：用于分割的阈值，取值在 0-255 之间，例如当threshold = 100，那么图片上小于 100 的转为 0，大于等于 100 的转为gray值

(int)gray：取值在 0-255 之间，通常取值为 255，当灰度图片中的灰度值大于threshold时，转换为gray值

(int)type：分割类型，处理完后，type=0 代表白色背景黑色物体，type=1 代表黑色背景白色物体

【返回值】

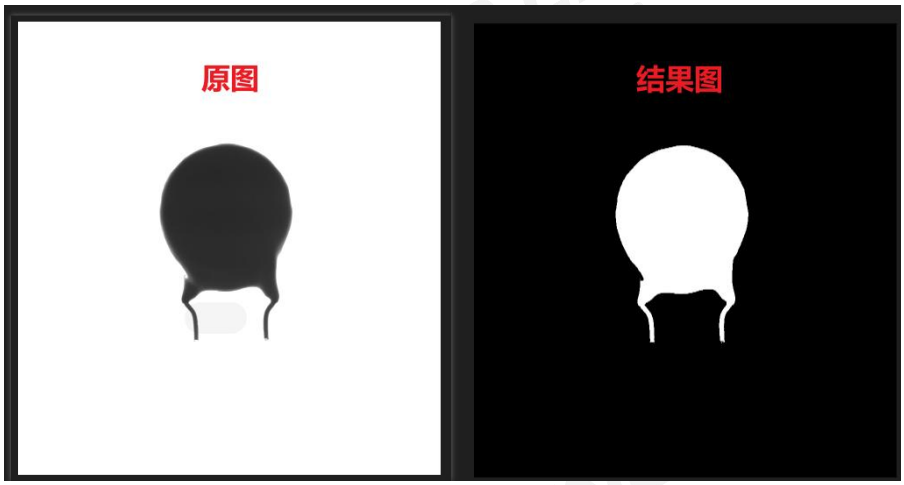
(array)运行成功返回处理后的二值图，运行失败返回空

【使用案例】

```
var image = HVisionModule.readImage("/home/test.bmp", 0)
var thrImage = HVisionModule.thresholdProcessed(image, 200, 255, 0)
var flag = HVisionModule.saveImage(thrImage , "/home/thr.bmp")
print("保存flag = ",flag)
flag = HVisionModule.clearImage(image)
flag = HVisionModule.clearImage(thrImage )
```

【执行效果】

保存 flag = true



3.2.23 开运算

【接口】

```
var openImage = HVisionModule.openProcessed(image, xStep, yStep)
```

【描述】

开运算是图像处理中的一种形态学操作，通常用于去除图像中的小物体或噪声。首先对图像

进行腐蚀操作，这会使图像中的物体缩小，并可能使它们分离或消失。然后再进行膨胀操作，这会使图像中的物体重新增大，但是腐蚀操作时去除的小物体或噪声已经被消除，就无法再被增大，因此先腐蚀再膨胀后，可以消除一些小的物体或噪声。

【参数】

(array)image：输入图片变量

(int)xStep：x 方向的处理步长，xStep 越大，x 方向的处理作用越强

(int)yStep：y 方向的处理步长，yStep 越大，y 方向的处理作用越强

【返回值】

(array)运行成功返回处理后的图片，运行失败返回空

【使用案例】

```
var image = HVisionModule.readImage("/home/test.bmp", 0)
var thrImage = HVisionModule.thresholdProcessed(image, 200, 255, 0)
var openImage = HVisionModule.openProcessed(thrImage, 10, 10)
var flag = HVisionModule.saveImage(openImage , "/home/open.bmp")
print("保存flag = ",flag)
flag = HVisionModule.clearImage(image)
flag = HVisionModule.clearImage(thrImage )
flag = HVisionModule.clearImage(openImage )
```

【执行效果】

保存 flag = true

3.2.24 图片相减

【接口】

```
var subImage = HVisionModule.imageSubtraction(image1, image2)
```

【描述】

灰度图片相减

【参数】

(array)image1：输入图片变量 1

(array)image2：输入图片变量 2

【返回值】

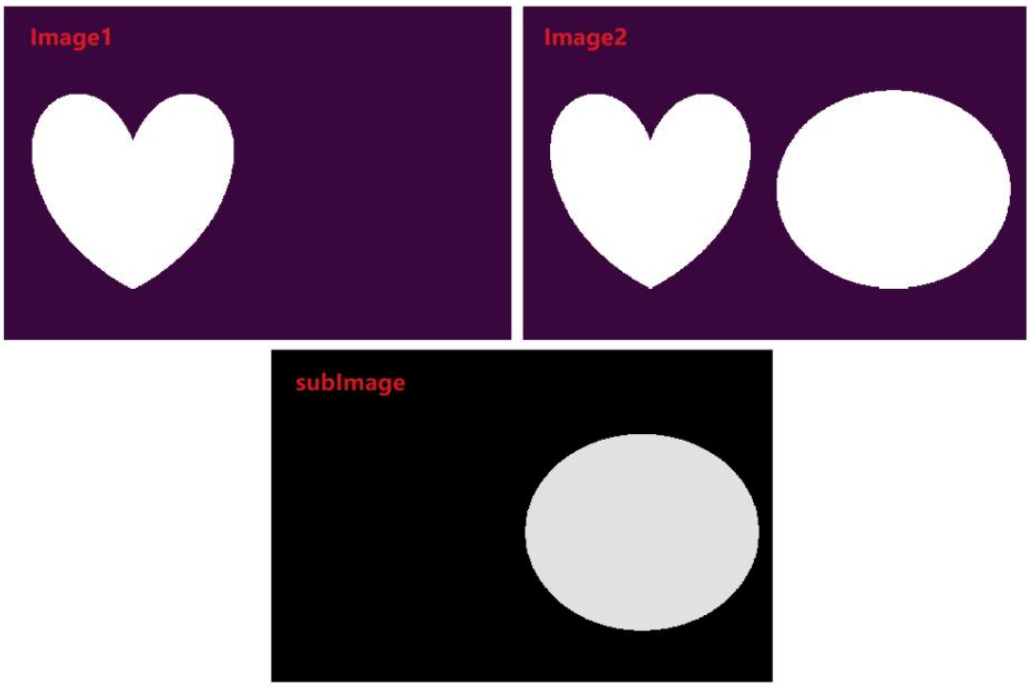
(array)运行成功返回相减后的图片，运行失败返回空

【使用案例】

```
var image1 = HVisionModule.readImage("/home/Image1.bmp", 0)
var image2 = HVisionModule.readImage("/home/Image2.bmp", 0)
var subImage = HVisionModule.imageSubtraction(image1 , image2 )
var flag = HVisionModule.saveImage(subImage , "/home/sub.bmp")
print("保存flag = ",flag)
flag = HVisionModule.clearImage(Image2)
flag = HVisionModule.clearImage(Image1)
flag = HVisionModule.clearImage(subImage)
```

【执行效果】

```
保存 flag = true
```



3.2.25 获取图片的所有物体的正矩形列表

【接口】

var rectList = HVisionModule.getImageObjectRect(image)

【描述】

输入二值图，返回图片上所有白色的区域（矩形框[x,y,w,h]）列表

【参数】

(array)image：输入图片变量，最好是二值图，否则难以保证寻找效果

【返回值】

(array)运行成功返回区域列表如:[[x1,y1,w1,h1],[x2,y2,w2,h2]]，运行失败返回空

【使用案例】

```
var image = HVisionModule.readImage("/home/test.bmp", 0)
var thrImage = HVisionModule.thresholdProcessed(image, 100, 255, 0)
var rectList = HVisionModule.getImageObjectRect(thrImage)
flag = HVisionModule.clearImage(image)
```

```
flag = HVisionModule.clearImage(thrImage)
```

【执行效果】

待补充

3.2.26 获取图片平均哈希值

【接口】

```
var strHashList = HVisionModule.averageHashFun(image, listRect)
```

【描述】

输入图片和区域,使用平均哈希算法计算图片上各个区域的哈希值,并返回一个字符串列表,用于对比图片中各个区域的相似度

【参数】

(array)image：输入图片变量

(array)listRect：输入处理的区域列表，通常由getImageObjectRect()函数获取

【返回值】

(array)运行成功返回图片的各个区域的哈希值组成的列表，运行失败返回空

【使用案例】

```
var image = HVisionModule.readImage("/home/test.bmp", 0)
var thrImage = HVisionModule.thresholdProcessed(image, 100, 255, 0)
var rectList = HVisionModule.getImageObjectRect(thrImage)
var strHashList = HVisionModule.averageHashFun(image, rectList )
flag = HVisionModule.clearImage(image)
flag = HVisionModule.clearImage(thrImage)
```

【执行效果】

哈希值：1111111111111101111010111011011011011100000000000000100000000000001000000000000000.....

3.2.27 对比图片哈希值

【接口】

var HVisionModule.contrastHashFun(str1, str2)

【描述】

比较两个哈希值列表，列表的大小需要相等，通常比较由两张不同的图片使用同一个区域列表计算的哈希列表，0 为最小值，400*列表大小为最大值

【参数】

(array)str1：哈希值列表 1

(array)str2：哈希值列表 2

【返回值】

(int)运行成功返回对比值，对比值越小越相似，运行失败返回-1

【使用案例】

```
var image = HVisionModule.readImage("/home/test.bmp", 0)
var image2 = HVisionModule.readImage("/home/test2.bmp", 0)
var thrImage = HVisionModule.thresholdProcessed(image, 100, 255, 0)
var rectList = HVisionModule.getImageObjectRect(thrImage)
var strHashList = HVisionModule.averageHashFun(image, rectList)
var strHashList2 = HVisionModule.averageHashFun(image2, rectList)
var value = HVisionModule.contrastHashFun(strHashList, strHashList2)
print("对比值为：", value)
flag = HVisionModule.clearImage(image)
flag = HVisionModule.clearImage(image2)
flag = HVisionModule.clearImage(thrImage)
```

【执行效果】

```
对比值为：46
```

3.2.28 保存图片

【接口】

```
var flag = HVisionModule.saveImage(image, path)
```

【描述】

保存图片到本地

【参数】

(array)image：输入需要保存的图片变量

(string)path：输入保存的路径和名字如path = "/home/usr/123.bmp"

【返回值】

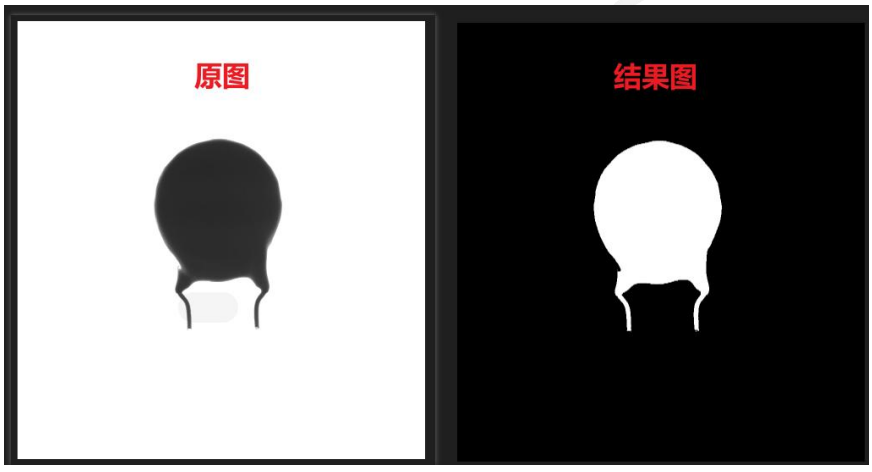
(bool)运行成功返回true，运行失败返回false

【使用案例】

```
var image = HVisionModule.readImage("/home/test.bmp", 0)
var thrImage = HVisionModule.thresholdProcessed(image, 200, 255, 0)
var flag = HVisionModule.saveImage(thrImage , "/home/thr.bmp")
print("保存flag = ",flag)
flag = HVisionModule.clearImage(image)
flag = HVisionModule.clearImage(thrImage )
```

【执行效果】

保存 flag = true



3.2.29 清理图片内存

【接口】

var flag = HVisionModule.clearImage(image)

【描述】

手动清理图片内存。在所有接口中，如果返回值为图片类型的，在使用完之后，都需要手动清除图片内存

【参数】

(array)image：输入图片变量，清理该图片内存。

【返回值】

(bool)运行成功返回true，运行失败返回false

【使用案例】

```
var image = HVisionModule.readImage("/home/test.bmp", 0)  //以灰度图片读入
var imageColor = HVisionModule.cvtColor(image , 1)          //转成彩色图片
var flag = HVisionModule.clearImage(image)
print("清理灰度图片内存flag = ", flag)
var flag2 = HVisionModule.clearImage(imageColor)
print("清理图片内存flag2 = ", flag2)
```

【执行效果】

```
清理灰度图片内存 flag = true
清理图片内存 flag2 = true
```

3.2.30 设置绘制字体大小

【接口】

var flag = HVisionModule.setTextSize(image, size)

【描述】

设置所有绘制的字体大小

【参数】

(array)image：输入图片，指定该图片当前绘制字体的大小

(int)size：字体的大小，默认值为 5，无单位，通过调试设置该值

【返回值】

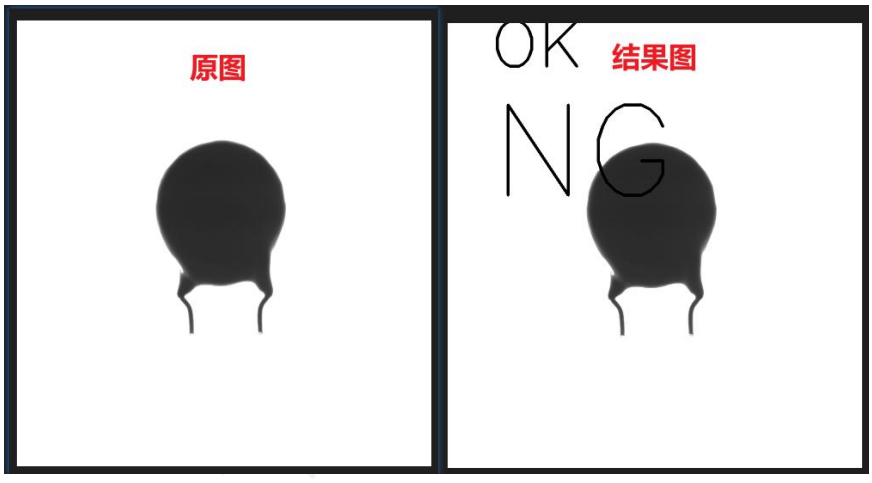
(bool)运行成功返回true，运行失败返回false

【使用案例】

```
var image = HVisionModule.readImage("/home/test.bmp", 0)
flag = HVisionModule.setTextSize(image, 5)
var flag = HVisionModule.drawText(image, "OK", 100, 100)
flag = HVisionModule.setTextSize(image, 10)
var flag = HVisionModule.drawText(image, "NG", 100, 400)
flag = HVisionModule.saveImage(image, "/home/text.bmp")
print("保存flag = ",flag)
flag = HVisionModule.clearImage(image)
```

【执行效果】

保存 flag = true



3.2.31 边缘检测

【接口】

var image = HVisionModule.edgeDetect(image, highThreshold, lowThreshold, coreSize)

【描述】

边缘检测，输入灰度图片，调整检测参数，返回图片中的边缘图

【参数】

(array)image: 输入图片变量

(int)highThreshold: 高阈值，高于该数值，认为是边缘，取值为 0-255

(int)lowThreshold: 低阈值，低于该阈值，不被认为是边缘，取值 0-255，一般为 highThreshold的 1/2 左右

(int)coreSize: 边缘计算核心大小，一般设置为 3，值越大，检测的边缘需要越明显

【返回值】

(array)运行成功返回image的边缘的图片，运行失败返回空。

【使用案例】

```
var image = HVisionModule.readImage("/home/test.bmp", 0)
var edgeimage = HVisionModule.edgeDetect(image , 200, 150, 3)
flag = HVisionModule.saveImage(edgeimage , "/home/edge.bmp")
print("保存flag = ",flag)
var flag1 = HVisionModule.clearImage(edgeimage )
var flag2 = HVisionModule.clearImage(image)
```

【执行效果】

```
保存 flag = true
```

3.2.32 获取拟合直线

【接口】

varlineList = HVisionModule.getFittedLine(image, thr, objectColor, polarityFlag, polarityDirection, edgeSize)

【描述】

获取图片中拟合的直线。

【参数】

(array)image：输入图片变量

(int)thr：查找图片中轮廓点的阈值，使用轮廓点拟合直线，范围为-1~255，其中-1 为自动阈值

(bool)objectColor：阈值分割后的目标颜色，0 为黑白反转，1 为保持黑白不变，默认值为 0

(bool)polarityFlag：直线极性，0 为从白到黑，1 为从黑到白，默认值为 0

(int)polarityDirection：直线极性方向，0 为从上到下，1 为从左到右，默认值为 0，默认从上到下

(int)edgeSize：直线拟合时的边缘搜索大小，只允许填写三个值：3、5、7，默认值为 5。通常来说，如果处理的是高分辨率图像或者需要更精确的边缘检测，可以考虑增大孔径大小

【返回值】

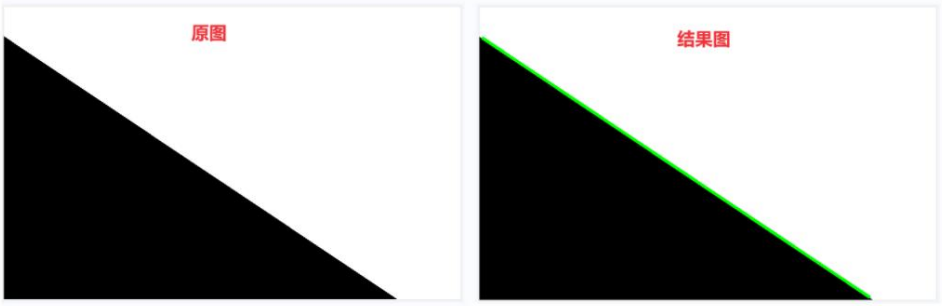
(array)返回一个数组result[x1,y1,x2,y2,k,b]，其中x1、y1、x2、y2 为直线的起点和终点坐标（以线段的形式表现），k、b为直线方程中y=kx+b中的k、b。多条直线时结果数组为多条直线的数据，每条直线 6 个数据。

【使用案例】

```
var image = HVisionModule.readImage("/home/test.bmp", 0)
var lineList = HVisionModule.getFittedLine(image, 100)
print(lineList)
//绘制彩色效果图
var colorimg = HVisionModule.cvtColor(image,1)
HVisionModule.setPenColor(colorimg,0,255,0)
var drawline = HVisionModule.drawLine(colorimg,lineList[0],lineList[1],lineList[2],lineList[3])
var flag1 = HVisionModule.clearImage(colorimg )
var flag2 = HVisionModule.clearImage(image)
```

【执行效果】

```
[x1, y1, x2, y2, k, b] = [5, 51, 642, 477, 0.668056309, 48.5107421875]
```



3.2.33 拟合直线的相对坐标转换

【接口】

```
var axisList = HVisionModule.lineConvertByOrigin(lineList, offsetX, offsetY)
```

【描述】

把在裁剪图中的拟合直线坐标转换为原图的拟合直线坐标

【参数】

(array)lineList：拟合直线相对裁剪图的坐标数组

(int)offsetX：X方向偏移量，为裁剪图ROI的左上角坐标X

(int)offsetY：Y方向偏移量，为裁剪图ROI的左上角坐标Y

【返回值】

(array)成功返回拟合直线相对原图的坐标数组，失败返回空数组

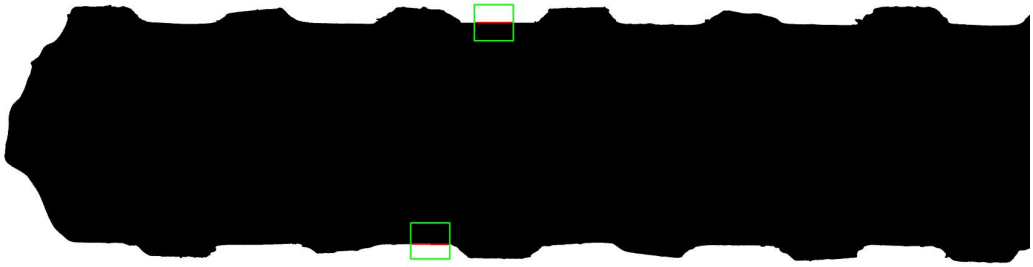
【使用案例】

```
var image = HVisionModule.readImage("/home/root/hust/05.bmp", 0)     //读取图片
var reduceImg1 = HVisionModule.cutImage(image1,1600,520,130,120)  //裁剪需拟合直线位置
var reduceImg2 = HVisionModule.cutImage(image1,1390,1240,130,120)
var lineList1 = HVisionModule.getFittedLine(reduceImg1, 150)  //拟合直线
var lineList2 = HVisionModule.getFittedLine(reduceImg2, 150)
var originList1 = HVisionModule.lineConvertByOrigin(lineList1,1600,520)    //相对坐标转换
var originList2 = HVisionModule.lineConvertByOrigin(lineList2,1390,1240)
HVisionModule.clearImage(reduceImg1)
HVisionModule.clearImage(reduceImg2)
HVisionModule.clearImage(image)
```

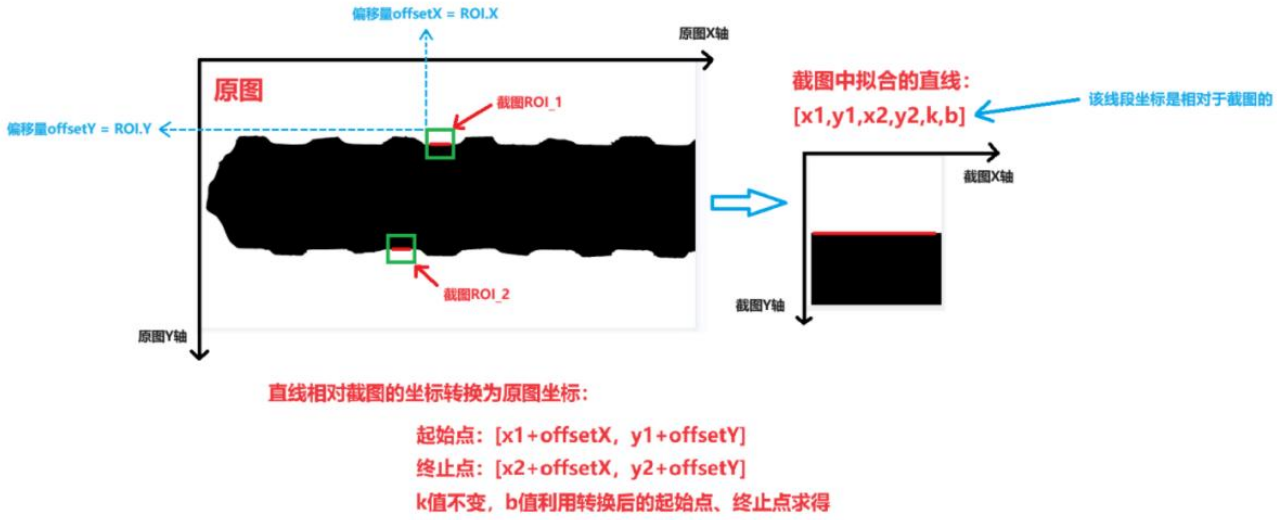
【执行效果】

```
lineList1:[x1, y1, x2, y2, k, b]= [5, 59, 125, 59, 0, 59]
originList1:[x1, y1, x2, y2, k, b]=[1605, 579, 1725, 579, 0, 579]

lineList2:[x1, y1, x2, y2, k, b]=[5, 70, 125, 71, 0.00040645, 70.96531677]
originList2:[x1, y1, x2, y2, k, b]=[1395, 1310, 1515, 1311, 0 00040645, 1309 .90861282]
```



相对坐标转换解析：



3.2.34 获取直线距离

【接口】

```
var distance = HVisionModule.getLineDistance(lineList1, lineList2, error)
```

【描述】

获取两条平行直线的距离

【参数】

(array)lineList1：输入直线 1 的数组[x1,y1,x2,y2,k,b]

(array)lineList2：输入直线 2 的数组[x1,y1,x2,y2,k,b]

(double)error：直线 1 斜率与直线 2 斜率的允许误差区间，当(k1-k2)∈[-error,error]，认为两条直线平行，默认为 0

【返回值】

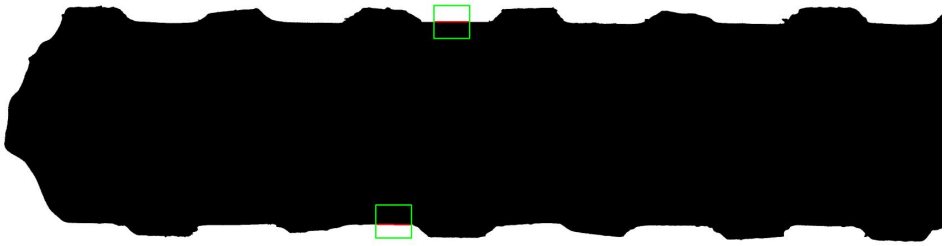
(double)运行成功返回两条平行直线的距离，运行失败返回-1

【使用案例】

```
var image = HVisionModule.readImage("/home/root/hust/05.bmp", 0)     //读取图片
var reduceImg1 = HVisionModule.cutImage(image1,1600,520,130,120)  //裁剪需拟合直线位置
var reduceImg2 = HVisionModule.cutImage(image1,1390,1240,130,120)
var lineList1 = HVisionModule.getFittedLine(reduceImg1, 150)  //拟合直线
var lineList2 = HVisionModule.getFittedLine(reduceImg2, 150)
var originList1 = HVisionModule.lineConvertByOrigin(lineList1,1600,520)    //相对坐标转换
var originList2 = HVisionModule.lineConvertByOrigin(lineList2,1390,1240)
var distance = HVisionModule.getLineDistance(originList1,originList2,0.2); //计算两条直线距离
HVisionModule.clearImage(reduceImg1)
HVisionModule.clearImage(reduceImg2)
HVisionModule.clearImage(image)
```

【执行效果】

两条直线的距离为：730.9086128218623



3.2.35 获取直线夹角

【接口】

var distance = HVisionModule.getLineAngle(slope1, slope2, error)

【描述】

获取两条直线的夹角

【参数】

(array)slope1：输入直线 1 的斜率

(array)slope2：输入直线 2 的斜率

(double)error：直线 1 斜率与直线 2 斜率的允许误差区间，当(slope1-slope2)∈[-error,error]，认为两条直线平行，默认为 0

【返回值】

(double)运行成功返回两条直线的夹角，运行失败返回 0

【使用案例】

```
var image = HVisionModule.readImage("/home/root/hust/05.bmp", 0)     //读取图片
var reduceImg1 = HVisionModule.cutImage(image1,1725,520,75,120)
var reduceImg2 = HVisionModule.cutImage(image1,1800,530,45,45)
var lineList1 = HVisionModule.getFittedLine(reduceImg1, 150)
```

```
var lineList2 = HVisionModule.getFittedLine(reduceImg2, 150)
HVisionModule.getLineAngle(lineList1[4],lineList2[4],0);
HVisionModule.clearImage(reduceImg1)
HVisionModule.clearImage(reduceImg2)
HVisionModule.clearImage(image1)
```

【执行效果】

两条直线的夹角为：45°

3.2.36 图片数据转换

【接口】

```
var imageData = HVisionModule.imageToData(image)
```

【描述】

把图片变量转换成可直接显示的数据，数据无需清理内存

【参数】

(array)image：输入图片变量

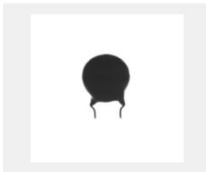
【返回值】

(array)返回一个数据变量，可以直接用于动态图片插件的显示

【使用案例】

```
var image = HVisionModule.readImage("/home/root/hust/05.bmp", 0)     //读取图片
var data = HVisionModule.imageToData(image)       //图片转成内存格式
Form.dynamicImage.loadFromData(data)              //动态图片插件加载显示图片
HVisionModule.clearImage(image)
```

【执行效果】



3.2.37 获取白色区域面积

【接口】

var areaList= HVisionModule.getObjectArea(image)

【描述】

计算图片中白色区域的面积，注意：输入图片需为阈值分割后的图像

【参数】

(array)image：输入图片变量

【返回值】

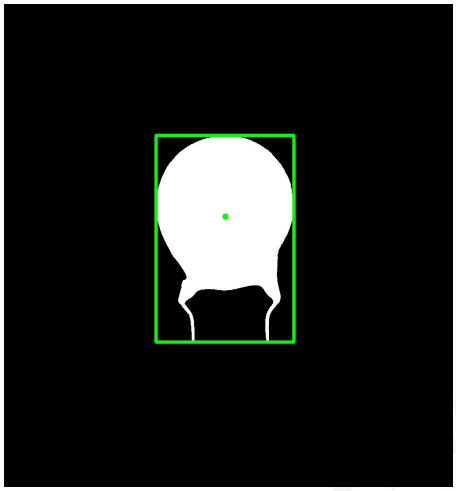
(array)返回一个数组result[area,centerX,centerY]，其中area为白色区域的面积，centerX、centerY分别为白色区域的中心点坐标X、Y

【使用案例】

```
var image = HVisionModule.readImage("/home/root/hust/05.bmp", 0)     //读取图片
var thrImg = HVisionModule.thresholdProcessed(image,150,255,1)       //图片阈值分割
var areaList= HVisionModule.getObjectArea(thrImg)       //计算图片白色区域面积
print(“白色区域面积为:”+areaList[0])
var contours = HVisionModule.getMaxContours(thrImg,150,1)
var rectList = HVisionModule.getBoundingRect(contours)
```

【执行效果】

```
白色区域面积为：81311
```



3.2.38 求线段与轮廓的切点及距离

【接口】

var resultList= HVisionModule.lineToContours(x1, y1, x2, y2, contours);

【描述】

获取线段的斜率，并计算得到该斜率与轮廓的切点以及切点到线段的距离（最远的相切点）

【参数】

(int)x1：线段的起始点坐标X1

(int)y1：线段的起始点坐标Y1

(int)x2：线段的终止点坐标X2

(int)y2：线段的终止点坐标Y2

(array)contours：输入的轮廓点集

【返回值】

(array)返回一个数组result[pointX,pointY,distance]，其中pointX、pointY为切点的坐标X、Y，distance为切点到线段的距离。

【使用案例】

```
var image = HVisionModule.readImage("/home/root/hust/05.bmp", 0)  //读取图片
var thrImg = HVisionModule.thresholdProcessed(image,150,255,1)      //图片阈值分割
```

```
var contours = HVisionModule.getMaxContours(thrImg ,100,1)      //获取最大轮廓
//开运算处理得到外接矩形或拟合直线的目标
var openimg = HVisionModule.openProcessed(thrImg,210,1)
var rectContours= HVisionModule.getMaxContours(openimg,100,1)      //获取拟合矩形轮廓
//通过外接矩形或者拟合直线得到线段的起始点、终止点
var list = HVisionModule.getMinAreaRect(rectContours)             //拟合外接矩形
var points =HVisionModule.rectToPoints(ContoursList)                  //矩形转换为顶点坐标
contours = HVisionModule.getRangeContour(contours ,1,0,points[1])  //以y方向筛选轮廓
//以斜矩形上边长为基准线获取切点与切点到线段的距离
var result1 = HVisionModule.lineToContours(points[0], points[1], points[6], points[7], contours)
//以斜矩形右边长为基准线获取切点与切点到线段的距离
var result2 = HVisionModule.lineToContours(points[4], points[5], points[6], points[7], contours)
print(“切点 1 到斜矩形上边长的距离:”, result1[2])
print(“切点 2 到斜矩形右边长的距离:”, result2[2])
HVisionModule.clearImage(thrImg)
HVisionModule.clearImage(openimg)
HVisionModule.clearImage(image)
```

【执行效果】

```
切点 1 到斜矩形上边长的距离：288.48663798
切点 2 到斜矩形右边长的距离：415.69551668
```

3.2.39 模板匹配

【接口】

var resultList= HVisionModule.getMatchTemplate(image,tempImage,type,score)

【描述】

对图片进行模板匹配，并返回匹配结果

【参数】

(array)image：输入待进行模板匹配的图片变量

(array)tempImage：输入标准模板的图片变量

(int)type：输入模板匹配的计算类型，目前仅允许输入三种类型：1、3、5;其中类型 1：标准平方差匹配；类型 3：标准相关性匹配，采用模板和图像间的乘法操作；类型 5：相关性系数匹配，将模版对其均值的相对值与图像对其均值的相关值进行匹配

(float)score：输入模板匹配的得分标准，分数为 0~100（模板匹配得分越高，说明匹配区域与标准模板越相似）

【返回值】

(array)返回一个数组result[topX,topY,maxValue]，topX、topY分别为匹配区域的左上角坐标X、Y，maxValue为匹配的最大得分，若maxValue小于score时，返回数组[0,0,0]

【使用案例】

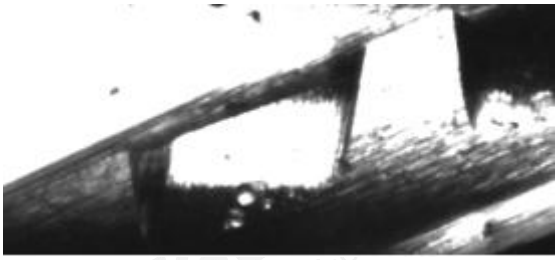
```
var image = HVisionModule.readImage("/home/root/usr/image.bmp", 0)  //读取图片
var rectinfo = Form.hVisionView.getItemInfo(-1)  //在视觉插件中编辑矩形后获取矩形信息
```

```
var xTop = rectinfo[1] - rectinfo[3] / 2
var yTop = rectinfo[2] - rectinfo[4] / 2
var w = rectinfo[3]
var h = rectinfo[4]
var reduceImg = HVisionModule.cutImage(image, xTop, yTop, w, h)  //裁图作为模板使用
//也可以直接读取模板图片
//var tempImage= HVisionModule.readImage("/home/root/usr/image.bmp", 0)
var resultList= HVisionModule.getMatchTemplate(iamge, reduceImg, 3, 90)  //模板匹配
var colorImg = HVisionModule.cvtColor(image,1)
HVisionModule.setPenColor(colorImg,0,255,0)
HVisionModule.drawRectangle(colorImg, xtop, ytop, w, h)  //绘制矩形显示

HVisionModule.clearImage(colorImg)
HVisionModule.clearImage(reduceImg)
HVisionModule.clearImage(iamge)
```

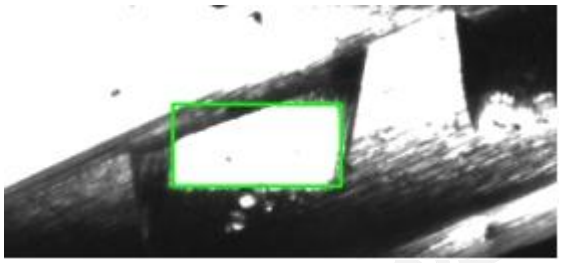
【执行效果】

原图：



模板：

效果图：



3.2.40 图像缩放

【接口】

var resizeImg= HVisionModule.resizeImage(image,sizeX, sizeY)

【描述】

将图像按比例进行放大或缩小

【参数】

(array)image：输入图片变量

(float)sizeX：输入x方向的比例缩放因子

(float)sizeY：输入y方向的比例缩放因子

【返回值】

(array)运行成功返回缩放后的图片变量，运行失败返回为空

【使用案例】

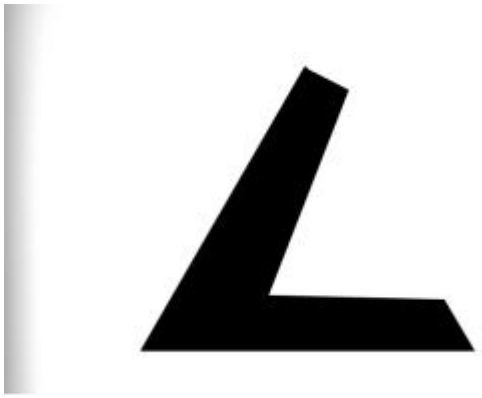
```
var image = HVisionModule.readImage("/home/root/hust/05.bmp", 0)     //读取图片
var resizeImg1 = HVisionModule.resizeImage(image,0.25,0.25)
var resizeImg2 = HVisionModule.resizeImage(image,1.25,1.25)
HVisionModule.saveImage(resizeImg,"/home/root/hust/resizeImg1.bmp") //存图查看缩放效果
HVisionModule.saveImage(resizeImg,"/home/root/hust/resizeImg2.bmp")
HVisionModule.clearImage(resizeImg1)
HVisionModule.clearImage(resizeImg2)
HVisionModule.clearImage(image )
```

【执行效果】

原图：

缩放因子(0.25,0.25)：

缩放因子(1.25,1.25)：



3.2.41 图像旋转

【接口】

var rotateImg= HVisionModule.rotateImage(image,rotateAngle)

【描述】

将图像按逆时针方向旋转

【参数】

(array)image：输入图片变量

(int)rotateAngle：输入图片的旋转角度，图片逆时针旋转，只允许填写 90，180，270；若旋转角度为 90 或 270，图像宽高也会改变；

【返回值】

(array)运行成功返回旋转后的图片变量，运行失败返回为空

【使用案例】

```
var image = HVisionModule.readImage("/home/root/hust/05.bmp", 0)     //读取图片
var rotateImg = HVisionModule.rotateImage(image,90)
HVisionModule.saveImage(rotateImg,"/home/root/hust/rotateImg.bmp") //存图查看旋转效果
HVisionModule.clearImage(rotateImg)
HVisionModule.clearImage(image)
```

【执行效果】

原图：

逆时针旋转 90 度：

4. 注意事项以及使用建议

1. bmp图片大小的计算方式为：（假设 4000*3000 的分辨率）

- 灰度图片：4000*3000/1024/1024 = 11.4MB

- 彩色图片：4000*3000*3/1024/1024 = 34.2MB

2. 图片传输耗时理论值计算：（假设 11.4MB大小的图片，从相机端到设备端）

- 百兆网口/百兆网线（传输速度 12.5MB/s）：11.4/12.5 = 912ms
- 千兆网口（传输速度 125MB/s）：11.4/125 = 91.2ms
- USB3.0 接口（传输速度 375MB/s）:11.4/375 = 30.4ms

3. 在高速的项目中，建议使用中低分辨率的相机、大帧率相机、黑白相机、USB3.0 接口、低曝光时间、硬触发方式，以减少相机在采集图片、传输图片时所耗费的时间。
4. 在控制器的系统中，可以使用变数、IO、Interrupt等触发方式控制HMI执行宏，在宏中执行视觉的相关功能。其中Interrupt的触发方式是效率最高的。

5. 免责声明

本文档提供有关 FINGER CNC 系列产品的信息，本文档未授予任何知识产权的许可，并未以明示或暗示，或以禁止发言或其它方式授予任何知识产权许可。除在其产品的销售条款和条件声明的责任之外，我公司概不承担任何其它责任。并且，我公司对本产品的销售和/或使用不作任何明示或暗示的担保，包括对产品的特定用途适用性、适销性或对任何专利权、版权或其它知识产权的侵权责任等均不作担保。未经本公司书面许可均构成侵权，一经发现，本公司将依法追究侵权人的法律责任。本公司有权随时对产品规格及产品描述做出修改且无需另行通知。

FINGER CNC

广州亿达科技有限公司

咨询热线：020-39389901　　维修专线：18127931302

传真号码：020-39389903　　邮政编码：511495

电子邮箱：finger@fingercnc.com

公司官网：www.finger-cnc.com

公司地址：广东省广州市番禺区钟村街诚鼎街 8 号 1 楼

亿达官网

亿达微信公众号