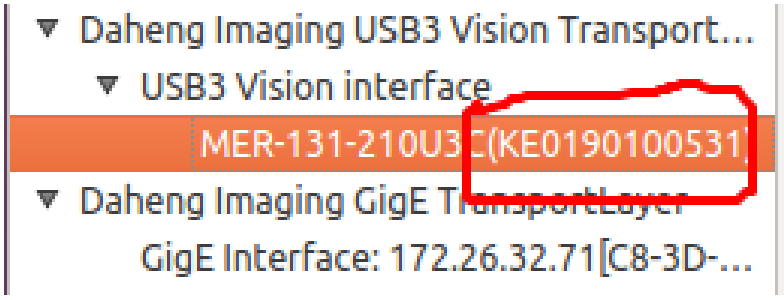
大恒双目

通过序列号判断左右相机

1. **直接打开大恒的官方驱动里面的GUI界面**

/Galaxy_Linux-x86_Gige-U3_32bits-64bits_1.2.1911.9122/Galaxy_camera/bin/GalaxyView



红圈部分就是我们需要的相机的序列号，是根据每个相机生产是的序号来确定的，每个相机都是惟一的。

2. **在打开设备的时候选择序列号打开**

```
#ifdef TINDEX_FSN
    stOpenParam.openMode    = GX_OPEN_INDEX;
#else
    stOpenParam.openMode    = GX_OPEN_SN;
#endif
```

通常我们默认是用序号（GX_OPEN_INDEX），改为序列号（GX_OPEN_SN）

驱动双相机

只初始化相机的库一遍，但是打开设备的时候打开两个

```
// 分别根据序列号打开两个设备
stOpenParam.accessMode = GX_ACCESS_EXCLUSIVE;
stOpenParam.openMode = GX_OPEN_SN;
stOpenParam.pszContent = "NE0160050005";
status = GXOpenDevice(&stOpenParam, &hDevice1);
stOpenParam.pszContent = "NE0160080024";
status = GXOpenDevice(&stOpenParam, &hDevice2);


if (status == GX_STATUS_SUCCESS)
{
    //两个相机都设置采集模式为连续采集
    status = GXSetEnum(hDevice1, GX_ENUM_ACQUISITION_MODE, GX_ACQ_MODE_CONTINUOUS);
    status = GXSetEnum(hDevice2, GX_ENUM_ACQUISITION_MODE, GX_ACQ_MODE_CONTINUOUS);

    //两个相机都 SET_EXPLOTURE
    double val = EXPLOSURETIME;
    status = GXSetFloat(hDevice1, GX_FLOAT_EXPOSURE_TIME, val);
    status = GXSetFloat(hDevice1, GX_FLOAT_EXPOSURE_TIME, val);
    cout << "主相机设置后曝光值: " << val << endl;

    ////////////////////// add ////////////////////////
    // 两个相机都设置分辨率
    status = GXSetInt(hDevice1, GX_INT_HEIGHT, VIDEO_HEIGHT);
    status = GXSetInt(hDevice1, GX_INT_WIDTH , VIDEO_WIDTH );
```

```
26      status = GXSetInt(hDevice2, GX_INT_HEIGHT,
VIDEO_HEIGHT);
27      status = GXSetInt(hDevice2, GX_INT_WIDTH ,
VIDEO_WIDTH );
28
29      //两个相机都设置触发开关为OFF
30      status = GXSetEnum(hDevice1,
GX_ENUM_TRIGGER_MODE, GX_TRIGGER_MODE_OFF);
31      status = GXSetEnum(hDevice2,
GX_ENUM_TRIGGER_MODE, GX_TRIGGER_MODE_OFF);
32
33      //两个相机都注册图像处理回调函数
34      status = GXRegisterCaptureCallback(hDevice1,
NULL, OnFrameCallbackFun1);
35      status = GXRegisterCaptureCallback(hDevice2,
NULL, OnFrameCallbackFun2);
36
37      //两个相机都发送开采命令
38      GXSendCommand(hDevice1,
GX_COMMAND_ACQUISITION_START);
39      GXSendCommand(hDevice2,
GX_COMMAND_ACQUISITION_START);
40
```

多线程采集和处理

实际上我们使用的是三线程，两个线程采集图片，一个线程处理图片

1. 定义全局的线程锁

```
volatile unsigned int prdIdx1;
volatile unsigned int prdIdx2;
volatile unsigned int csmIdx;

// 定义全局结构体，用于存放图片
struct ImageData {
    Mat img;
    unsigned int frame;
};
ImageData data1[BUFFER_SIZE];
ImageData data2[BUFFER_SIZE];
```

2. 在左相机的回调函数中

```
while (prdIdx1 - csmIdx >= BUFFER_SIZE);
data1[prdIdx1 % BUFFER_SIZE].img = src;
prdIdx1++;
```

3. 在右相机的回调函数中

```
while (prdIdx2 - csmIdx >= BUFFER_SIZE);
data2[prdIdx2 % BUFFER_SIZE].img = src;
prdIdx2++;
```

4. 在处理图片的线程中

```
while(prdIdx1 - csmIdx == 0 || prdIdx2 - csmIdx == 0);
data1[csmIdx % BUFFER_SIZE].img.copyTo(src1);
data2[csmIdx % BUFFER_SIZE].img.copyTo(src2);
++csmIdx;
if(src1.empty() || src2.empty()){
    cout<<"src empty"<<endl;
    continue;
}
```