



拼接 产线标定库

API 参考

文档版本 00B04

发布日期 2018-06-15

版权所有 © 深圳市海思半导体有限公司 2018。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



HISILICON、海思和其他海思商标均为深圳市海思半导体有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受海思公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，海思公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

深圳市海思半导体有限公司

地址：深圳市龙岗区坂田华为基地华为电气生产中心 邮编：518129

网址：<http://www.hisilicon.com>

客户服务电话：+86-755-28788858

客户服务传真：+86-755-28357515

客户服务邮箱：support@hisilicon.com



前 言

概述

拼接标定库是一套快捷、高性能的运行在 Windows 系统的软件库。标定库内部提供产线标定接口，并将产线标定的数据转换为单板可以直接使用的 LUT。标定库内部完成了标定的主要流程，并对外提供了灵活简单的 API，用户可以快速地开发应用程序，用于多路拼接产品的标定。



说明

- 未有特殊说明，Hi3559CV100 与 Hi3559AV100 内容一致。
- 未有特殊说明，Hi3556AV100 与 Hi3519AV100 内容一致。

产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本
Hi3559A	V100ES
Hi3559A	V100
Hi3559C	V100
Hi3519A	V100
Hi3556A	V100

读者对象






本文档（本指南）主要适用于以下工程师：

- 技术支持工程师
- 软件开发工程师



符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
 危险	表示有高度潜在危险，如果不能避免，会导致人员死亡或严重伤害。
 警告	表示有中度或低度潜在危险，如果不能避免，可能导致人员轻微或中等伤害。
 注意	表示有潜在风险，如果忽视这些文本，可能导致设备损坏、数据丢失、设备性能降低或不可预知的结果。
 窍门	表示能帮助您解决某个问题或节省您的时间。
 说明	表示是正文的附加信息，是对正文的强调和补充。

修订记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

文档版本 00B04 (2018-06-15)

第 4 次临时版本发布。

3.2 小节，HI_AVS_CAL_STITCH_MEASUREMENT_S【成员】涉及修改

4.3.2 小节，表 4-3 涉及修改

新增 4.3.3.4 小节

2.3、2.4、4.1.3 和 4.3.3 小节涉及修改

文档版本 00B03 (2018-05-15)

第 3 次临时版本发布。

1.1 小节，表 1-2 涉及修改

4.1.2、4.2.1.5、4.2.1.6 和 4.2.2.2 小节涉及修改

4.1.3 小节，图 4-3 涉及修改

新增第 5 章



文档版本 00B02 (2018-04-04)

第 2 次临时版本发布。

4.3.1 小节，表 4-2 涉及修改

4.3.2.2 小节新增鱼眼镜头标定数据采集的特殊要求，标定数据采集要求涉及修改

文档版本 00B01 (2018-01-31)

第 1 次临时版本发布。



目 录

前 言.....	i
1 概述.....	1
1.1 功能描述.....	1
1.2 函数描述方式.....	2
1.3 函数列表.....	3
1.4 结构体描述方式.....	3
2 API 函数说明	4
2.1 hiAVS_Version.....	4
2.2 hiAVS_CalProd	5
2.3 hiAVS_LutFromCalibration	7
2.4 hiAVS_PolyFromCalibration	10
3 数据类型与数据结构	13
3.1 通用数据类型.....	13
3.2 数据结构.....	14
4 开发指南.....	18
4.1 拼接标定应用总体方案介绍	18
4.1.1 整体标定方案.....	18
4.1.2 产线标定的组网环境.....	19
4.1.3 产线标定库内部具体使用流程.....	19
4.2 开发准备.....	20
4.2.1 下载和使用 OpenCV	20
4.2.2 产线标定库的开发.....	22
4.3 使用拼接标定库.....	24
4.3.1 运行拼接标定库.....	24
4.3.2 使用拼接标定库.....	24
4.3.3 第三方 (PTGui、hugin) 标定.....	32
4.3.4 LUT 及 BBox 表生成	35
5 API 应用实例	38
5.1 标定库应用流程.....	38



5.2 一个完整的 Sample 示例	38
5.3 Sample 编译运行的依赖条件简述	41



插图目录

图 2-1 产线标定接口 hiAVS_CalProd 处理流程	6
图 2-2 产线标定接口 hiAVS_CalProd 输入示例	6
图 2-3 LUT 表输入示例	9
图 4-1 整体标定方案	18
图 4-2 产线标定组网	19
图 4-3 拼接标定库的使用流程	20
图 4-4 产线标定在拼接标定流程中所处的环节	25
图 4-5 产线标定环境示意图	26
图 4-6 双鱼眼结构产线标定环境示意图	26
图 4-7 鱼镜头产线标定前需要配置 RadialCROP	27
图 4-8 Pqtools RadialCROP 的配置	28
图 4-9 边缘模糊，需要配置 RadialCROP 的图像	29
图 4-10 不同半径的 RadialCROP 效果预览	30
图 4-11 第三方标定在标定流程中所处的环节	33
图 4-12 PTGui 标定的主要流程	33
图 4-13 导出 PTGui 标定结果	34
图 4-14 使用 hugin 快速标定的步骤	34
图 4-15 LUT/Bbox 表生成在标定流程中所处的环节	35
图 4-16 Mask 图像示例	36



表格目录

表 1-1 拼接标定库开发包组件.....	1
表 1-2 拼接标定库开发运行环境.....	2
表 1-3 函数描述方式	2
表 1-4 函数列表	3
表 1-5 结构体描述方式	3
表 4-1 VS2015 project 常用属性配置表.....	23
表 4-2 标定库运行时依赖的动态库列表.....	24
表 4-3 产线标定结果的判断表.....	32
表 4-4 BboxLUT 的差异	32
表 4-5 Bbox 和 LUT 的差别	35



1 概述

1.1 功能描述

拼接标定库是一套快捷、高性能的运行在 Windows 系统的软件库。标定库内部提供产线标定，并将产线标定的数据转换为单板可以直接使用的 LUT，标定库内部完成了标定的主要流程，并对外提供了灵活简单的 API，用户可以快速地开发应用程序，用于多路拼接产品的标定。

拼接标定库可以为用户提供 Windows 环境下的动态库和静态库两种调用形式，可极大地方便地开发应用程序。标定库的主要组件及相关说明如表 1-1 所示。

表1-1 拼接标定库开发包组件

组件	名称	说明
API 接口	hi_type.h hi_avs_prod_calib.h	用户工程中，应该保证 先包含 hi_type.h，再包含 hi_avs_prod_calib.h 。
静态库	HiAVSCalibrationLib_release.lib	用户开发时，需要在工程中链接 HiAVSCalibrationLib_release.lib，以 Visual Studio 2015 为例（以下简称: VS2015），需要在 Project/Properties/Linker/Input 中添加 HiAVSCalibrationLib_release.lib；
动态库	HiAVSCalibrationLib_release.lib HiAVSCalibrationLib_release.dll	用户开发时，需要在工程中链接 HiAVSCalibrationLib_release.lib（添加方式和静态库相同）； 编译完成后，将 HiAVSCalibrationLib_release.dll 放在执行目录或者系统目录才能执行；
其他	OpenCV3.4.0	需要额外依赖于第三方的开源开发库，参考下载地址： http://opencv.org ；不包含在发布包中，需用户自行下载编译；



用户可在多种编译环境上进行基于拼接标定库的应用程序开发，兼容微软公司的 Windows 7 或更高版本的主流视窗操作系统，兼容 Intel 公司和 AMD 公司推出的绝大部分面向 PC 机的主流 CPU 芯片组。其主要开发以及运行环境说明如表 1-2 所示。

表1-2 拼接标定库开发运行环境

分类	兼容配置	推荐配置	说明
编译器	Visual Studio 系列	Visual Studio 2015	OpenCV3.4.0 只能使用 VS2015、VS2017 或更高版本（以 OpenCV 官方说明为准）
操作系统	Windows 7 Windows 10	Windows 7、 Windows10	由于高性能的需要，拼接产线标定库仅支持 x64 系统
硬件	Intel Core 系列 AMD Athlon 系列	高性能工作站	无

1.2 函数描述方式

表1-3 函数描述方式

参数域	作用
描述	简要描述 API 的主要功能。
语法	列出 API 的语法样式。
描述	简要描述 API 的工作过程。
参数	列出 API 的参数、参数说明及参数属性。
返回值	列出 API 的返回值及返回值说明。
错误码	描述与函数相关的错误码。
需求	描述函数的头文件依赖。
注意	使用 API 时应注意的事项。
举例	简易 Sample 代码实现。
相关主题	列出函数有关的结构体及函数。



1.3 函数列表

表1-4 函数列表

函数	功能	页码
hiAVS_Version	获取 AVS 标定库的版本号。	2.1
hiAVS_CalProd	产线标定算法接口。	2.2
hiAVS_LutFromCalibration	根据生成的标定文件转成可以直接使用的 LUT 表。	2.3
hiAVS_PolyFromCalibration	根据生成的标定文件转成边界表，仅 Hi3559AV100ES 需要额外使用到该边界表，其它芯片则不需要使用该接口。	2.4

1.4 结构体描述方式

表1-5 结构体描述方式

参数域	作用
说明	简要描述结构体所实现的功能。
定义	列出结构体的定义。
成员	列出结构体内的所有成员及描述。
注意事项	列出结构体的注意事项。
相关类型及接口	列出与该数据结构有关的其他类型与接口。



2 API 函数说明

2.1 hiAVS_Version

【描述】

获取 AVS 标定库的版本号。

【语法】

```
HI_S32 hiAVS_Version(HI_CHAR cAVSCalibVersion[128]);
```

【参数】

参数名称	描述	输入/输出
cAVSCalibVersion	存放获取的版本号	输入

【返回值】

返回值	描述
0	获取版本号成功
非 0	获取失败

【错误码】

接口返回值	含义
HI_SUCCESS	返回成功
HI_FAILURE	返回失败

【需求】



- 头文件: hi_type.h、hi_avs_prod_calib.h
- 库文件: HiAVSCalibrationLib_release.lib

【注意】

需要先包含 hi_type.h, 再包含 hi_avs_prod_calib.h

【举例】

```
HI_CHAR cAVSCalibVersion[128];  
hiAVS_Version(cAVSCalibVersion);  
cout << "Welcome to Hisi AVS Calibration " << cAVSCalibVersion << endl;
```

【相关主题】

无

2.2 hiAVS_CalProd

【描述】

产线标定算法接口。

【语法】

```
HI_AVS_STATUS_E hiAVS_CalProd(HI_U32 u32Camera, const HI_CHAR *  
pcInputCalFile, const HI_CHAR * pcOutputCalFile, HI_FLOAT fFixtureRadius,  
const HI_CHAR * pcCalImage[], HI_AVS_CAL_STITCH_MEASUREMENT_S &stResult);
```

【参数】

参数名称	描述	输入/输出
u32Camera	AVS 拼接镜头的数量,范围:[2,6]	输入
pcInputCalFile	根据镜头模型生成的标定文件 (.cal 文件)	输入
pcOutputCalFile	产线标定算法输出的标定文件 (.cal 文件)	输出
fFixtureRadius	产线标定环境的半径 (长度单位: 米), 范围 [0.2, 20]	输入
pcCalImage	包含完整文件路径的产线标定图片, 其数量和 u32Camera 对应;	输入
stResult	用于评价算法标定情况的数据;	输出

【返回值】



返回值	描述
0	成功。
非 0	失败，具体查看错误码结构体 HI_AVS_STATUS_E

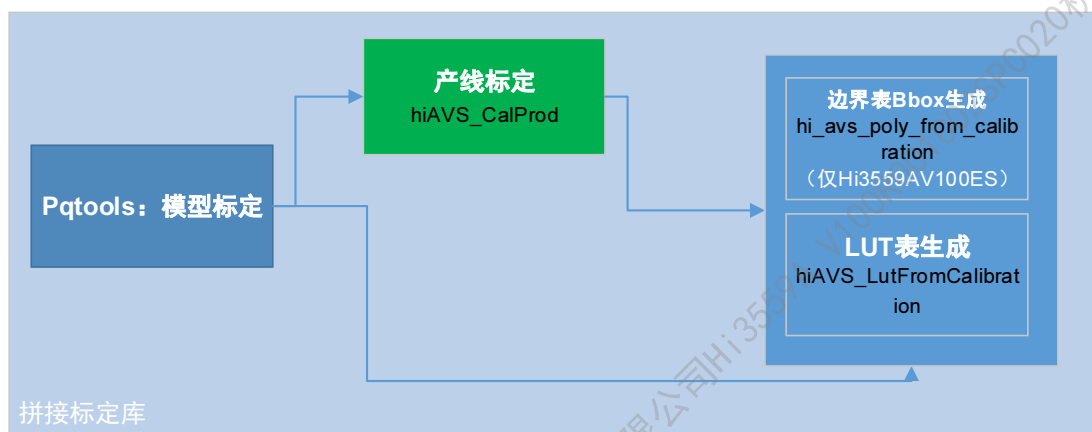
【需求】

头文件：hi_type.h、hi_avs_prod_calib.h

【注意】

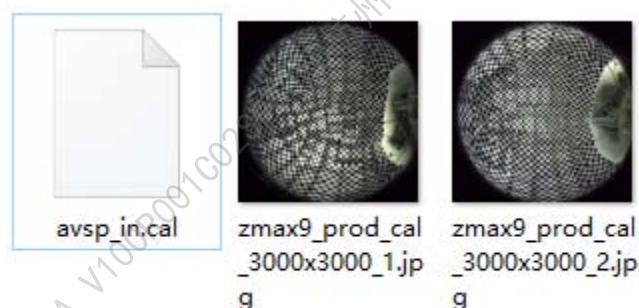
- 需要先包含 hi_type.h，再包含 hi_avs_prod_calib.h
- 该接口需要依赖于模型标定的输入，且标定结果需要转换才能应用，其处理流程如图 2-1 所示。

图2-1 产线标定接口 hiAVS_CalProd 处理流程



- 产线标定需要 模型标定文件、镜头重叠区的标定图片，如图 2-2 所示。

图2-2 产线标定接口 hiAVS_CalProd 输入示例



- 模型标定文件需要使用 PQtools 进行模型标定输出的标定文件，镜头重叠区的标定图片需要在特定环境下抓拍。



- 关于标定结果的评价，请参考 `HI_AVS_CAL_STITCH_MEASUREMENT_S`
- 产线标定后生成一个新的 .cal 标定文件，还需要调用 `hiAVS_LutFromCalibration` 或 `hiAVS_PolyFromCalibration`（仅 Hi3559AV100ES 需要）转换为 LUT 才能通过 SDK MPI 接口直接导入到相机设备中。

【举例】

```
const HI_CHAR * pcCalImage[2];
HI_AVS_CAL_STITCH_MEASUREMENT_S stResult;
pcCalImage[0] = "..\\..\\test\\for_prod_cal\\camera_0.jpg";
pcCalImage[1] = "..\\..\\test\\for_prod_cal\\camera_1.jpg";
HI_S32 s32Ret = hiAVS_CalProd(
    2,
    "..\\..\\test\\for_prod_cal\\prod_step1.cal",
    "..\\..\\test\\for_prod_cal\\prod_step1_V4.cal",
    (HI_FLOAT)1.1,
    pcCalImage,
    stResult);
cout << "ret = " << s32Ret << endl;
cout << "maximum_reproj_err: " << stResult.dMaxReprojErr << endl;
cout << "average_reproj_err: " << stResult.dAverageReprojErr << endl;
cout << "total_matched_points: " << stResult.dTotalMatchedPoints << endl;
```

【相关主题】

- [HI_AVS_CAL_STITCH_MEASUREMENT_S](#)
- [hiAVS_LutFromCalibration](#)
- [hiAVS_PolyFromCalibration](#)

2.3 hiAVS_LutFromCalibration

【描述】

根据生成的标定文件转成可以直接使用的 LUT 表，其标定来源有三个：模型标定¹、`hiAVS_CalProd`、第三方标定工具（PTgui、Hugin）。

【语法】

```
HI_AVS_STATUS_E hiAVS_LutFromCalibration(const HI_CHAR *
pcCalibrationFile, const HI_CHAR * pcMaskPrefix, const HI_CHAR *
pcOutputPrefix, HI_FLOAT fStitchDistance, HI_AVS_LUT_ACCURACY_E
enLutAccuracy, HI_BOOL bIsHiAvsCal);
```

¹ 模型标定可以使用 PQtools 生成；



【参数】

参数名称	描述	输入/输出
pcCalibrationFile	来自于模型标定、产线标定接口 hiAVS_CalProd 或第三方标定工具（Ptgui、Hugin）标定文件的路径。	输入
pcMaskPrefix	定义每个摄像头图像有效区域的掩码文件前缀。索引 n 处的照相机将使用带有文件名 prefix_n.png 的掩码文件。每个 mask 文件在前缀的后面添加 _n.png，索引编号从 0 开始。该掩码文件的分辨率大小应与此照相机的图像大小相同。掩码中的白色像素表示图像有效显示区域，而黑色像素表示不使用的区域。对于普通非鱼眼图像，mask 可以使用全白图像。对于鱼眼镜头，将使用和镜头有效像素相同覆盖的白色圆圈。Mask 图片只能使用.png 格式。	输入
pcOutputPrefix	每个镜头输出的 LUT 表文件前缀。每个镜头的文件名将以 prefix_n.bin 的形式存储（每个 LUT 表在前缀后面添加 _n.bin,n 为镜头数字编号，从 0 开始）。	输出
fStitchDistance	对海思标定结果（hiAVS_Cal 、hiAVS_CalProd）可以配置最佳拼接距离，根据此最佳拼接距离生成对应的 LUT 表（范围：>0.5，单位:米）； 对第三方标定，该参数无效，其标定效果由标定时决定。	输入
enLutAccuracy	LUT 的精度配置，只能配置为 Hi_AVS_LUT_ACCURACY_HIGH=0、 Hi_AVS_LUT_ACCURACY_LOW=1；高精度需要更高的性能，需要根据产品性能进行配置。	输入
bIsHiAvsCal	配置是否使用海思 AVS 标定库生成的的标定文件（来源于：hiAVS_Cal 、hiAVS_CalProd）；如果不使用海思标定库，则标定文件来自第三方标定结果（*.pto）。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，具体查看错误码结构体 HI_AVS_STATUS_E



【需求】

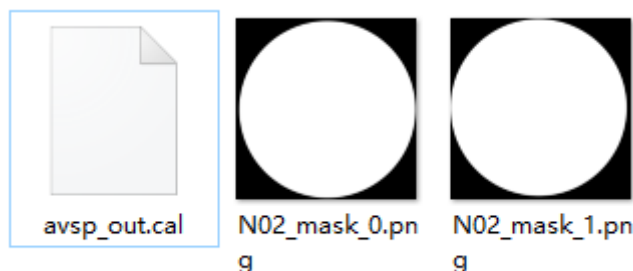
- 头文件：hi_type.h、hi_avs_prod_calib.h
- 库文件：HiAVSCalibrationLib_release.lib

【注意】

所有标定结果（包含模型标定、hiAVS_CalProd、第三方标定）都需要使用该接口将标定文件生成对应板端可以直接调用的 LUT 表。

该步骤下所需的文件如图 2-3 所示。

图2-3 LUT 表输入示例



注意

mask 需要配置完整的路径前缀，且 mask 目前只支持 PNG 文件，若有其他的图片格式，请先使用 图像处理工具（如 mspaint 画图、Photoshop...）打开，另存为 png 格式。Mask 文件的生成方式可以参考 4.3.4 “LUT 及 BBox 表生成”。

【举例】

```
HI_CHAR
cCalibrationFile[]="D:\\WORK\\AVS\\AVSPCalibrationLib\\PQT\\test\\for_mes
h_poly\\2fisheye.cal";
    HI_CHAR cMaskPrefix[] =
"D:\\WORK\\AVS\\AVSPCalibrationLib\\PQT\\test\\for_mesh_poly\\2fisheye_ma
sk";
    HI_CHAR
cOutputPrefix[]="D:\\WORK\\AVS\\AVSPCalibrationLib\\PQT\\test\\for_mesh_p
oly\\2fisheye_Poly";
    HI_FLOAT fStitchDistance    =    4;
    HI_AVS_LUT_ACCURACY_E enLutAccuracy    = HI_AVS_LUT_ACCURACY_HIGH;
    HI_BOOL bIsHiAvsCal= HI_TRUE;
    HI_S32 s32Ret =
hiAVS_LutFromCalibration(cCalibrationFile,cMaskPrefix,cOutputPrefix,fStit
chDistance,enLutAccuracy,bIsHiAvsCal);
    if (s32Ret != HI_SUCCESS)
```



```
{
    cout << "!!Error:_" << __LINE__ << "_", ret=" << s32Ret << endl;
    cout << "\t\tcCalibrationFile:" << cCalibrationFile << endl;
    cout << "\t\tcMaskPrefix:" << cMaskPrefix << endl;
    cout << "\t\tcOutputPrefix:" << cOutputPrefix << endl;
    cout << "\t\tfStitchDistance:" << fStitchDistance << endl;
    cout << "\t\tfenLutAccuracy:" << enLutAccuracy << endl;
    cout << "\t\tbIsHi_AvsCal:" << bIsHiAvsCal << endl;
}
else
{
    cout << "\tsucess" << endl;
}
```

【相关主题】

- [HI_AVS_STATUS_E](#)
- [HI_AVS_LUT_ACCURACY_E](#)
- [hiAVS_CalProd](#)

2.4 hiAVS_PolyFromCalibration

【描述】

根据生成的标定文件转成边界表，**仅 Hi3559AV100ES** 需要额外使用到该边界表，其它芯片不需要使用该接口。

【语法】

```
HI_AVS_STATUS_E hiAVS_PolyFromCalibration(const HI_CHAR *
pcCalibrationFile, const HI_CHAR * pcMaskPrefix, const HI_CHAR *
pcOutputPrefix, HI_FLOAT fStitchDistance, HI_BOOL bIsHiAvsCal);
```

【参数】

参数名称	描述	输入/输出
pcCalibrationFile	来自于模型标定、产线标定接口 hiAVS_CalProd 或 第三方标定工具标定文件的路径。	输入
pcMaskPrefix	定义每个摄像头图像有效区域的掩码文件前缀。索引 n 处的照相机将使用带有文件名 prefix_n.png 的掩码文件。每个 mask 文件在前缀的后面添加 _n.png，索引编号从 0 开始。该掩码文件的分辨率大小应与此照相机的图像大小相同。掩码中的白色像素表示图像有效显示区域，而黑色像素表示不使用的区域。对于普通非鱼眼图像，mask 可以使用全白图像。对于鱼镜头头，将使用和镜头有效像素相	输入



参数名称	描述	输入/输出
	同覆盖的白色圆圈。Mask 图片只能使用.png 格式。	
pcOutputPrefix	每个镜头输出的 LUT 表文件前缀。每个镜头的文件名将以 prefix_n.bin 的形式存储（每个 LUT 表在前缀后面添加 _n.bin,n 为镜头数字编号，从 0 开始）；	输出
fStitchDistance	对海思标定结果（模型标定、hiAVS_CalProd）可以配置最佳拼接距离，根据此最佳拼接距离生成对应的 LUT 表（范围：>0.5，单位:米）；对第三方标定，该参数无效，其标定效果由标定时决定。	输入
bIsHiAvsCal	配置是否使用海思 AVS 标定库生成的标定文件（来源于：模型标定、hiAVS_CalProd）；如果不使用海思标定库，则标定文件来自第三方标定结果（*.pto）。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，具体查看错误码结构体 HI_AVS_STATUS_E

【需求】

- 头文件：hi_type.h、hi_avs_prod_calib.h
- 库文件：HiAVSCalibrationLib_release.lib

【注意】

仅用于 Hi3559AV100ES。

【举例】

```
HI_CHAR
cCalibrationFile[]="D:\\WORK\\AVS\\AVSPCalibrationLib\\PQT\\test\\for_mes
h_poly\\2fisheye.cal";
HI_CHAR cMaskPrefix[] =
"D:\\WORK\\AVS\\AVSPCalibrationLib\\PQT\\test\\for_mesh_poly\\2fisheye_ma
sk";
HI_CHAR
cOutputPrefix[]="D:\\WORK\\AVS\\AVSPCalibrationLib\\PQT\\test\\for_mesh_p
oly\\2fisheye_Poly";
HI_FLOAT fStitchDistance    =    4;
HI_BOOL bIsHiAvsCal= HI_TRUE;
```



```
HI_S32 s32Ret =
hiAVS_PolyFromCalibration(cCalibrationFile,cMaskPrefix,cOutputPrefix,fSti
tchDistance,bIsHiAvsCal);
if (s32Ret != HI_SUCCESS)
{
    cout << "!!Error:_" << __LINE__ << "_", ret=" << s32Ret << endl;
    cout << "\t\tcCalibrationFile:" << cCalibrationFile << endl;
    cout << "\t\tcMaskPrefix:" << cMaskPrefix << endl;
    cout << "\t\tcOutputPrefix:" << cOutputPrefix << endl;
    cout << "\t\tfStitchDistance:" << fStitchDistance << endl;
    cout << "\t\tbIsHiAvsCal:" << bIsHiAvsCal << endl;
}
else
{
    cout << "\tsucess" << endl;
}
```

【相关主题】

- [HI_AVS_STATUS_E](#)
- [HI_AVS_LUT_ACCURACY_E](#)
- [hiAVS_CalProd](#)



3 数据类型与数据结构

3.1 通用数据类型

在 Windows 环境下，使用到的通用数据类型定义于 hi_type.h 中，调用接口前需要先包含 hi_type.h，具体定义如下：

```
typedef unsigned char      HI_U8;
typedef unsigned char      HI_UCHAR;
typedef unsigned short     HI_U16;
typedef unsigned int       HI_U32;
typedef signed char        HI_S8;
typedef short              HI_S16;
typedef int                HI_S32;
#ifndef _M_IX86
typedef unsigned long long HI_U64;
typedef long long          HI_S64;
#else
typedef __int64            HI_U64;
typedef __int64            HI_S64;
#endif
typedef char               HI_CHAR;
typedef char*              HI_PCHAR;
typedef float              HI_FLOAT;
typedef double             HI_DOUBLE;
typedef void               HI_VOID;
typedef unsigned long      HI_SIZE_T;
typedef unsigned long      HI_LENGTH_T;

/*-----*
 * const defination *
 *-----*/
typedef enum {
    HI_FALSE = 0,
```



```
        HI_TRUE      = 1,
    } HI_BOOL;

#ifndef NULL
#define NULL          0L
#endif
#define HI_NULL        0L
#define HI_NULL_PTR    0L
#define HI_SUCCESS     0
#define HI_FAILURE     (-1)
```

3.2 数据结构

- [HI_AVS_STATUS_E](#): AVS 接口返回状态。
- [HI_AVS_CAL_STITCH_MEASUREMENT_S](#): 用于评价算法标定结果的数据。
- [HI_AVS_LUT_ACCURACY_E](#): 枚举 LUT 精度。

HI_AVS_STATUS_E

【说明】

AVS 标定算法库接口返回状态。

【定义】

```
/**the status of avs interface returned*/
typedef enum hiAVS_STATUS_E
{
    HI_AVS_EOF = -1,      /*internal error codes*/
    HI_AVS_OK = 0,        /*success*/
    /* error statuses*/
    HI_AVS_UNABLE_TO_FIND_OVERLAP,
    HI_AVS_FILE_READ_ERROR,
    HI_AVS_FILE_WRITE_ERROR,
    HI_AVS_FILE_INVALID,
    HI_AVS_ALLOC_FAILED,
    HI_AVS_INVALID_PARAM,
    HI_AVS_STATUS_BUTT
}HI_AVS_STATUS_E;
```

【成员】

成员名称	描述
HI_AVS_EOF	=-1, 内部错误



成员名称	描述
HI_AVS_OK	=0, 函数执行成功
HI_AVS_UNABLE_TO_FIND_OVERLAP	无法找到重叠区域
HI_AVS_FILE_READ_ERROR	文件读取错误, 一般是图像宽、高等信息不匹配
HI_AVS_FILE_WRITE_ERROR	输出文件无法写入
HI_AVS_FILE_INVALID	文件不存在或路径错误
HI_AVS_ALLOC_FAILED	无法申请内存
HI_AVS_INVALID_PARAM	配置参数错误

【注意事项】

无

【相关数据类型及接口】

- [hiAVS_LutFromCalibration](#)
- [hiAVS_PolyFromCalibration](#)
- [hiAVS_CalProd](#)

HI_AVS_CAL_STITCH_MEASUREMENT_S

【说明】

用于评价算法标定结果的数据。

【定义】

```
typedef struct hiAVS_CAL_STITCH_MEASUREMENT_S
{
    HI_DOUBLE dMaxReprojErr;
    HI_DOUBLE dAverageReprojErr;
    HI_DOUBLE dTotalMatchedPoints;
}HI_AVS_CAL_STITCH_MEASUREMENT_S;
```

【成员】

成员名称	描述
dMaxReprojErr	最大标定反投影误差(以像素为单位), 用于判断标定过程中是否有棋盘格内角点匹配错误情况, 若该值大于 30 则需要检查标定文件是否符合要求 (如文件是否命名错误、图片中是否出现多个棋盘格、图片是否模糊等);



成员名称	描述
dAverageReprojErr	这是平均反投影错误 (以像素为单位), 若该值处于[0,1]范围, 标定效果最佳; 处于 (1, 2]范围, 标定效果满意; 处于 (2, 4]标定效果基本较好; 若大于 4, 则标定效果不理想。
dTotalMatchedPoints	指匹配成功的角点对, 与图像数量及棋盘格内角点数相关。

【注意事项】

标定结果的评估仅相对于已检测出角点的图片, 未检测出的无效图片不会影响结果的评估, 但它可能对最终的效果是有影响的。

【相关数据类型及接口】

[hiAVS_CalProd](#)

HI_AVS_LUT_ACCURACY_E

【说明】

定义 LUT 的精度。

【定义】

```
/**Specification of the LUT accuracy*/  
typedef enum hiAVS_LUT_ACCURACY_E  
{  
    HI_AVS_LUT_ACCURACY_HIGH = 0,  
    HI_AVS_LUT_ACCURACY_LOW = 1,  
    HI_AVS_LUT_ACCURACY_BUTT  
}HI_AVS_LUT_ACCURACY_E;
```

【成员】

成员名称	描述
HI_AVS_LUT_ACCURACY_HIGH = 0	高精度 LUT 表
HI_AVS_LUT_ACCURACY_LOW = 1	低精度 LUT 表
HI_AVS_LUT_ACCURACY_BUTT	保留字段

【注意事项】

LUT 表精度越高, 标定效果越好, 但相应的性能要求也越高。

【相关数据类型及接口】



[hiAVS_LutFromCalibration](#)

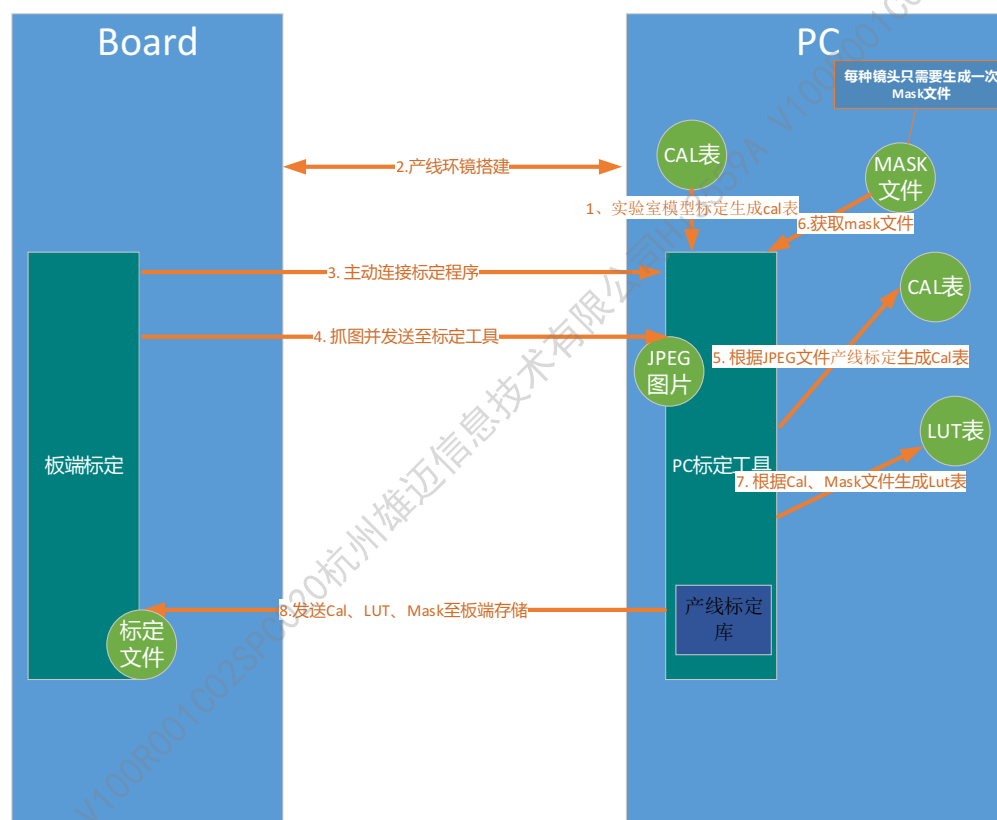


4 开发指南

4.1 拼接标定应用总体方案介绍

4.1.1 整体标定方案

图4-1 整体标定方案

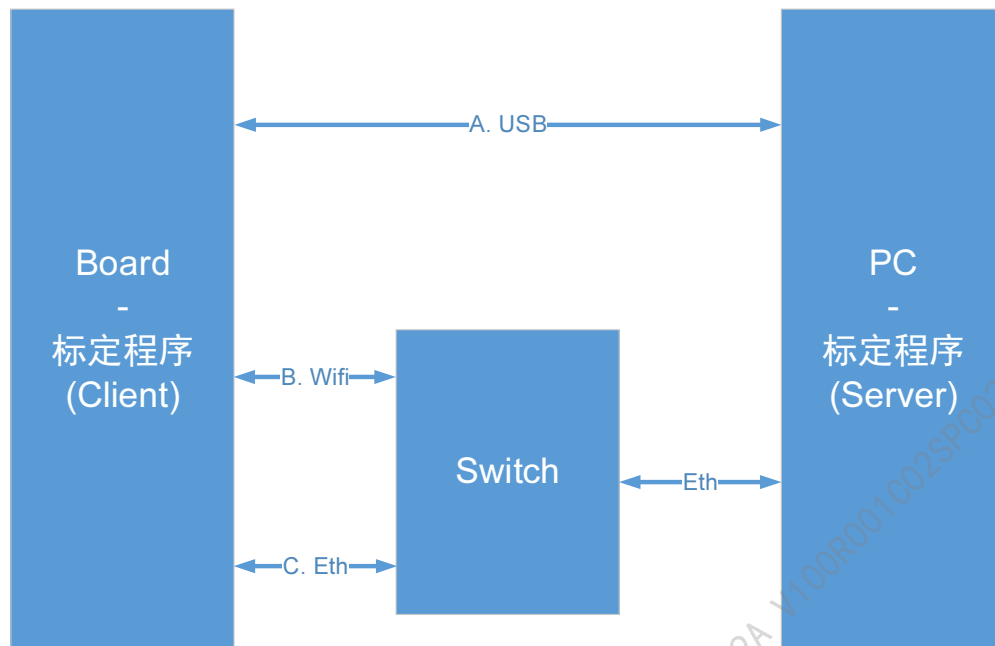




4.1.2 产线标定的组网环境

产线标定是针对设备个体进行，需要具备简单易用高效的方案，根据通讯接口的不同，可以使用 USB、无线网络、有线网络的一种或几种组合的组网方式开发标定应用。如图 4-2 所示。

图4-2 产线标定组网

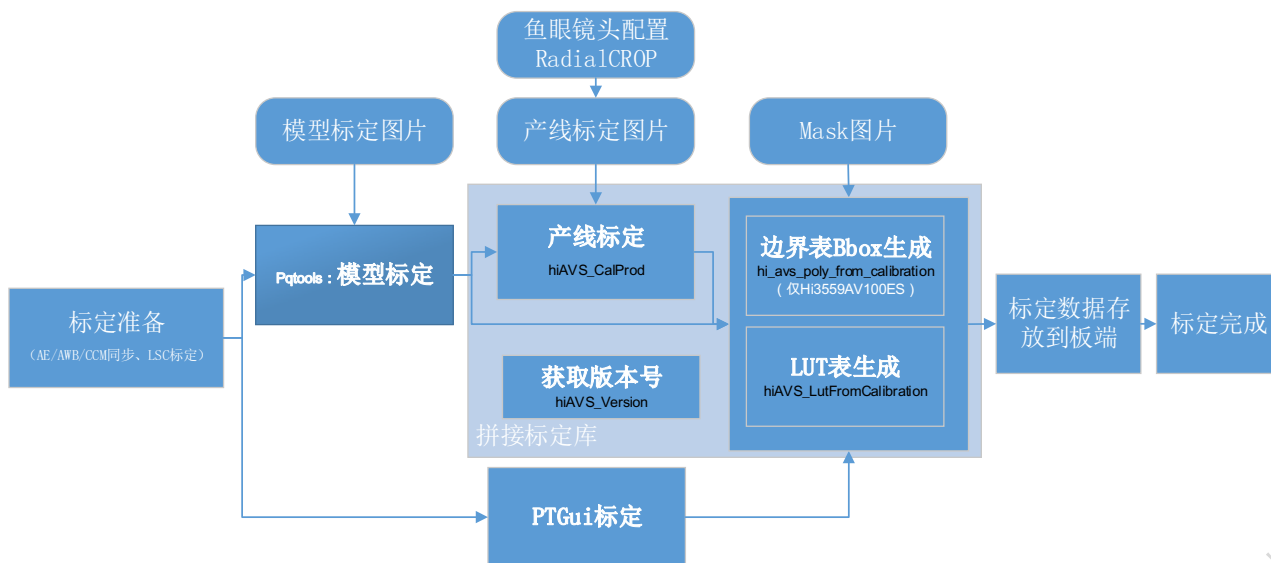


4.1.3 产线标定库内部具体使用流程

产线标定拼接标定需要在产品出厂前完成。标定前需要对每个镜头进行 AE/AWB/CCM 同步和 LSC 标定，且要做模型标定一次，在量产阶段还需要在每台设备出厂前进行一次产线标定，也可以将第三方标定结果应用于产品中，标定流程如图 4-3 所示。



图4-3 拼接标定库的使用流程



4.2 开发准备

拼接产线标定库需要额外使用 OpenCV 动态库，并且基于 windows 的环境开发和使
用。其主要开发过程需要包含 OpenCV 的编译及基于库文件的开发。

4.2.1 下载和使用 OpenCV

OpenCV 用户可以根据自己的需求进行定制化编译，这里仅提供一个可行的方案供参
考。

4.2.1.1 OpenCV 简介

OpenCV 是一个基于 BSD 许可（开源）发行的跨平台计算机视觉库，可以运行在
Linux、Windows、Android 和 Mac OS 操作系统上。它轻量级而且高效——由一系列 C
函数和少量 C++ 类构成，同时提供了 Python、Ruby、MATLAB 等语言的接口，实现
了图像处理 and 计算机视觉方面的很多通用算法。拼接产线标定库使用了 OpenCV 相
关的算法加速，因此需要运行时需要额外提供 OpenCV 动态库。

4.2.1.2 OpenCV 下载

OpenCV 属于第三方开源的库，可以在遵循 BSD 许可的情况下自由下载使用，下载地
址：<http://opencv.org>。具体下载和使用以官网为准，以下方式仅供参考。

在浏览器中输入官网地址：<http://opencv.org>；

当前使用的 OpenCV 版本是 3.4.0；

下载完成后选择一个路径直接解压 OpenCV。



4.2.1.3 CMake 工具简介

CMake 是一个跨平台的安装（编译）工具，可以用简单的语句来描述所有平台的安装（编译过程）。它能够输出各种各样的 makefile 或者 project 文件，能测试编译器所支持的 C++ 特性，类似 UNIX 下的 automake。这里使用 CMake 可以较方便的对 OpenCV 进行配置。

4.2.1.4 CMake 工具下载

请到官方网站 <https://cmake.org/> 下载最新版本安装即可。

4.2.1.5 使用 Cmake 配置 OpenCV 的编译环境

可以直接使用 cmake-gui.exe 配置 OpenCV 的编译环境，配置前请确保：

- OpenCV 已下载，并解压到指定目录；
- Cmake 已下载，并解压到指定目录；
- 已安装好任意一款 IDE 开发环境，如 VS2015；

Cmake-gui 的配置可以参考如下步骤：

步骤 1. 配置 source code （OpenCV 源代码所在路径）。

步骤 2. 配置 build （OpenCV 动态库生成路径）。

步骤 3. Configure: 配置 OpenCV，注意首次配置时需要选择 win32 还是 x64，这里提供的是 x64 开发库，只能选择对应 x64 的版本（如：Visual Studio 14 2015 Win64）。

步骤 4. Search: 如果第 3 步没有红色错误信息，则跳过这 4、5 步，如果有红色信息，需要确认该模块是否必要，如果非必要部分，请搜索到该模块，将其取消选择。以下可能出现的问题及处理方式仅供参考：

1. 对于 OpenCV3.4.0 来说，IPP 和 ffmpeg 需要在线下载然后进行配置，该下载地址可能无法下载，这里可以将其取消选择即可；
2. 若配置有 ffmpeg 报错信息，搜索 ffmpeg，去掉勾选，不使用在线下载以避免配置错误；
3. 若配置有 IPP 报错信息，搜索 IPP，去掉勾选，不使用在线下载以避免配置错误；

步骤 5. Configure: 如果第 3 步没有错误信息，则跳过这一步；否则需要重新配置一次。

步骤 6. Generate: 生成 vs 可编译的工程。

步骤 7. Open Project: 使用 vs 打开 Cmake 创建好的工程。

----结束

4.2.1.6 编译 OpenCV

本部分需要先完成如下条件：

- OpenCV 已下载，并解压到指定目录；
- Cmake 已下载，并且配置好 OpenCV；
- 已安装好任意一款 IDE 开发环境，如 VS2015；



以 VS2015 为例，介绍 OpenCV 的编译：

- 使用 VS2015 打开 OpenCV 配置好的 solution；
 - 在 CmakeGUI 界面直接点击 “Open Project” 打开项目；
 - 在之前配置好的 build 路径中，打开 OpenCV.sln；如：build/OpenCV.sln
- 选择生成 Release 库；
- ALL_BUILD: 在 CMakeTargets 下选择 ALL_BUILD，编译 OpenCV；
- INSTALL²: 在 CMakeTargets 下选择 INSTALL 通过 build 生成动态库到指定路径；
- 生成的 OpenCV 动态库的库路径：build\install\x64\vc14\lib、build\install\x64\vc14\bin；
- 开发时将 build\install\x64\vc14\lib 添加到开发项目中，并添加所需的库文件名称（HiAVSCalibrationLib 所需的 OpenCV 库有：
**opencv_core340.lib;opencv_imgproc340.lib;opencv_imgcodecs340.lib;
opencv_calib3d340.lib;** 由于版本发布库已链接了所需的 OpenCV 库文件，一般不需要在项目中再进行配置，只需要在运行时添加需要的 dll 即可）。
- 开发完毕后，需要拷贝对应的 OpenCV 动态库到执行路径或者系统库路径，需要的 OpenCV 动态库和链接的 lib 文件是相关的，当前分别是
opencv_core340.dll;opencv_calib3d340.dll;
opencv_imgcodecs340.dll ;opencv_imgproc340.dll; opencv_flann340.dll;
opencv_features2d340.dll

4.2.2 产线标定库的开发

产线标定库需要一个 Windows 上的 IDE 开发环境，这里以 VS2015 为例简单介绍工程项目的建立，步骤仅提供参考。

4.2.2.1 安装 IDE 开发环境

需要在 windows 上安装一个 IDE 开发环境，需要用户根据实际开发习惯选用 IDE。

4.2.2.2 使用 VS2015³创建开发环境

步骤 1. 创建一个工程

新建一个工程，可以将工程添加到现有 solution 或者新建一个 solution，最简单的是使用 win32 控制台的方式（当然也可以创建其他类型的工程，根据实际情况确定）；

如果已有源码，则要勾选上空项目，并选择控制台应用程序 Console application、动态库 dll、静态库任意其一。

步骤 2. 将源码添加到工程

直接右键点击创建好的项目，使用 Add\Existing Item 添加已有的源代码（包含源码和头文件）添加到项目或者新建文件进行编辑。

² ALL_BUILD 和 INSTALL，可以全部选中一次性 build（参考方法：按住键盘 CTRL，然后使用鼠标逐个点击选择）。

³ VS 2015 是一款商业软件，需用户自行购买，也可以使用其他版本的 IDE 开发工具替代。



步骤 3. 配置项目

需要将工程进行配置。项目配置较复杂，这里简单介绍最常用的部分。

表4-1 VS2015 project 常用属性配置表

配置项	参数	配置说明
Configuration	ALL Configurations、 debug、release	一般最常用的是配置成 release 就好了，或者配置为 ALL Configurations
Platform	Win32、x64	根据平台选择，当前只支持 x64
Configuration Type	Exe、dll、lib	开发工具，选 exe；动态库选 dll；静态库选 lib
Character Set	-	建议 Use Multi-Byte Character Set
Include Directories	-	配置头文件所在的路径，特别是代码中的源文件和头文件不在同一目录，而且 include 时不带路径时，编译会报错，这时要配置 头文件所在路径 ，使用绝对路径和相对路径都可以。 建议将发布包的 api 路径添加到这里。 注意最好不要删掉系统的配置，在前面增加自己的配置。
Library Directories	-	配置静态库或动态库（的 lib 文件）所在的路径 ，这里如果不配置，后面所有引用的库文件都要带路径，否则编译报错，找不到指定的库文件； 建议将 发布包里的 dll/lib 路径 添加到这里。
Additional Include Directories	-	配置额外的头文件路径，其作用可视为等同于 Include Directories，任选其一或者两个同时使用都可以。
Runtime Library	MT、MD、...	使用默认配置 MD。
Additional Dependencies	-	<ul style="list-style-type: none">需要连接的静态库或动态库（需要.lib 路径，暂不需要.dll），如果已经在 Library Directories 配置了路径，这里可以只写库名称，否则要完整的相对或绝对路径；路径配置错误，无法编辑，会提示对应的库文件找不到；需要将 HiAVSCalibrationLib_release.lib 配置到这里。

步骤 4. build

所有的准备就绪后，直接编译即可，注意编译平台（只能使用 x64）的配置。



----结束

4.3 使用拼接标定库

4.3.1 运行拼接标定库

运行需要额外的 OpenCV 动态库：

**opencv_core340.dll;opencv_calib3d340.dll;
opencv_imgcodecs340.dll ;opencv_imgproc340.dll; opencv_flann340.dll;
opencv_features2d340.dll,**

请将 OpenCV 动态库拷贝至系统目录或者当前执行目录。

如果是调用 AVS 标定库的动态库，还需要**将动态库 HiAVSCalibrationLib_release.dll 放至当前执行目录或系统目录。**

表4-2 标定库运行时依赖的动态库列表

分类	运行依赖的动态库
OpenCV 运行库	opencv_core340.dll;opencv_calib3d340.dll; opencv_imgcodecs340.dll ;opencv_imgproc340.dll; opencv_flann340.dll; opencv_features2d340.dll
AVS 标定库	HiAVSCalibrationLib_release.dll 如果使用静态库，则不需要额外的动态链接库
VS 运行库	标定库运行需要提供 VS2015 运行库（concr140.dll）,该动态库一般 vs2015 安装后可以在 C:\Windows\System32 找到。
其他	用户使用的第三方动态库。

4.3.2 使用拼接标定库

拼接标定库主要有三种使用方法：模型标定、产线标定、第三方标定。

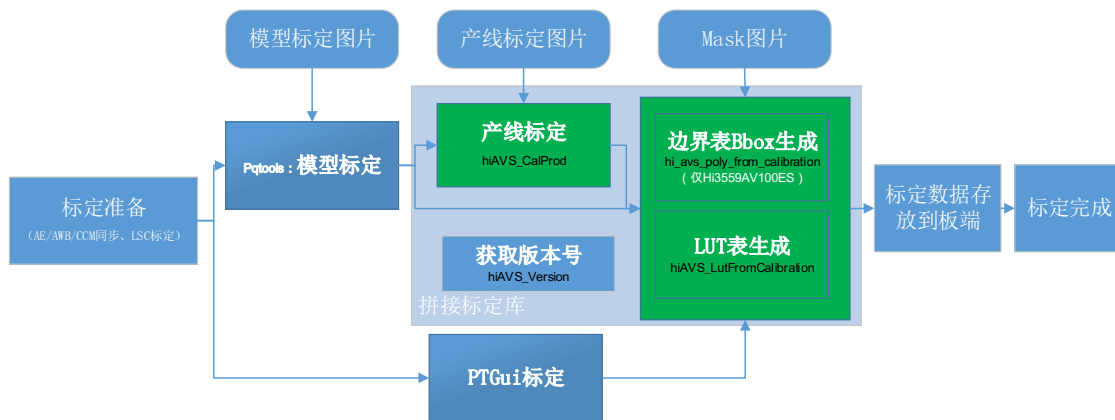
4.3.2.1 模型标定

模型标定需要参考文档《拼接调试指南》。



4.3.2.2 产线标定

图4-4 产线标定在拼接标定流程中所处的环节



产线标定主要是要针对拼接重叠区域进行标定，需要设计一个可以覆盖重叠区域的结构。

产线标定数据说明

- 产线标定不需要大量的标定图片，但是需要能覆盖镜头重叠区域的特定环境下抓拍的一组图片。
- 产线标定除了需要采集标定图片外，还需要 PQtools 上的模型标定的标定文件作为输入。

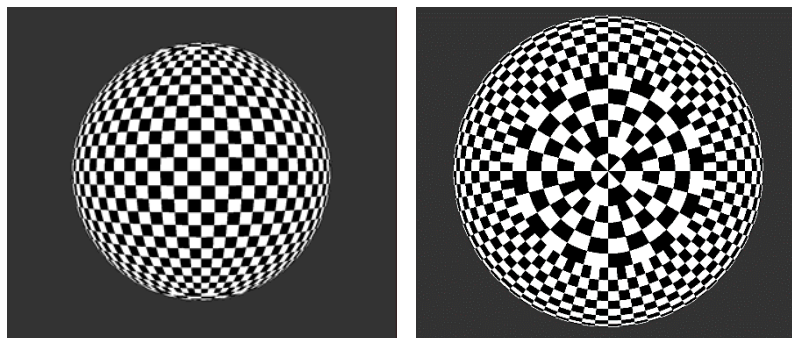
标定数据采集要求

产线标定阶段每个产品个体都需要标定，以解决个体之间的差异，标定时只需将个体放在特定球形环境里每路拍取一张图像即可。将拍取的图像与第一步生成的.cal 文件作为第二步输入，即可使用 AVS 标定工具生成针对该个体的标定结果.cal 文件。

产线标定需要一个较为严格的标定环境，理想环境为一个棋盘格球面，环境设计如图 4-5 所示。球体直径为 2 米，南北极之间有 36 个格子，赤道一周有 72 个格子，即一个格子 5° 。当然为了使格子均匀分布，南北极区域格子数相应减少，如图(b)所示。标定时全景相机放置在球体中心，由于是一个对称的球面，相机摆放在球心，方向无要求。



图4-5 产线标定环境示意图



(a) 正视图

(b) 俯视图



注意

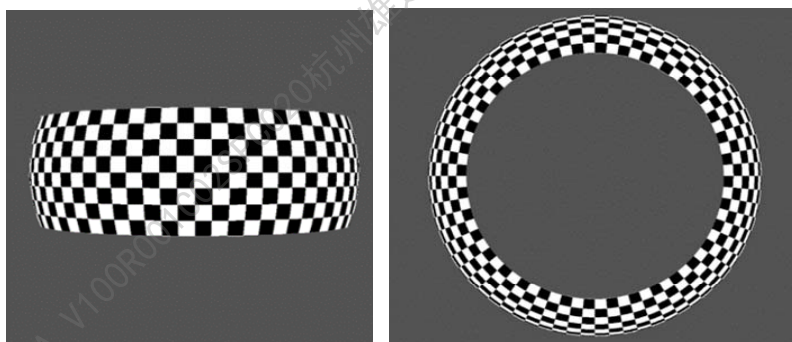
产线标定环境的大小与最佳拼接距离没有关联，标定完成后可在生成 LUT 时配置任意的最佳拼接距离。

技术上，可以对棋盘格和测试环境进行简化，简化后的棋盘格只需要覆盖成像区域即可，若是非全景相机，可根据产品形态及产线要求对理想标定环境进行修正，并保证棋盘格为球弧面分布，格子分布符合合适。以双鱼眼为特例说明。

由于双鱼眼结构的重叠区为环形，故可将球面两侧区域裁剪掉，保留赤道附近环球型即可。每个格子依旧为 5° ，故环形一周仍然是 72 个格子，垂直方向格子数量根据重叠区大小设计即可，比如镜头 FOV 为 200° ，则有 40° 重叠区，此时垂直方向需要 8 个格子以上，上下预留 1 个格子，10 个格子较为合适。

生产时需保证背景（即非棋盘格覆盖区域）干净，无其他棋盘格图样，避免标定时误检，匹配错误，造成标定失败。

图4-6 双鱼眼结构产线标定环境示意图



(a) 正视图

(b) 俯视图



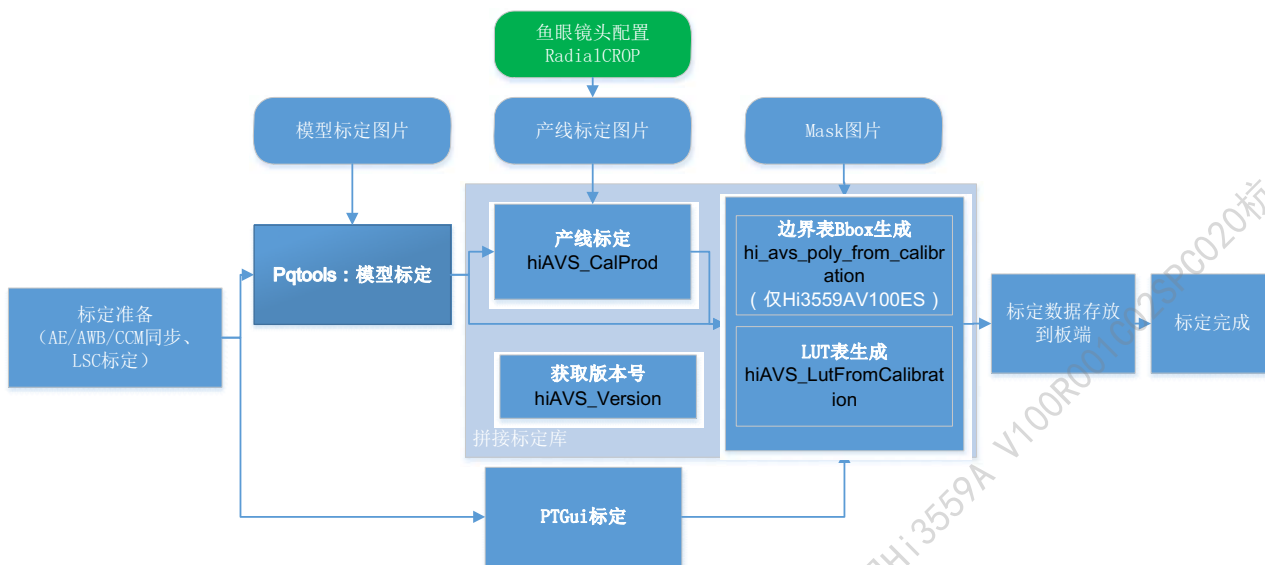
为降低产线标定时存在的干扰，镜头画面范围内尽量避免出现纹理细节复杂的物品，如地毯、花草等，保持背景区域简单可以有助于提升标定速度和标定效果。

鱼眼镜头标定数据采集的特殊要求

针对鱼眼镜头，由于靠近镜头边缘（如图 4-7 所示红色线条和绿色线条之间的部分）存在图像模糊、畸变达到最大这些因素的影响，可能导致产线标定误差增大，从而影响效果。为提升效果，需要方案上进行一些配合。

RadialCROP 需要在产线标定图片采集前配置。

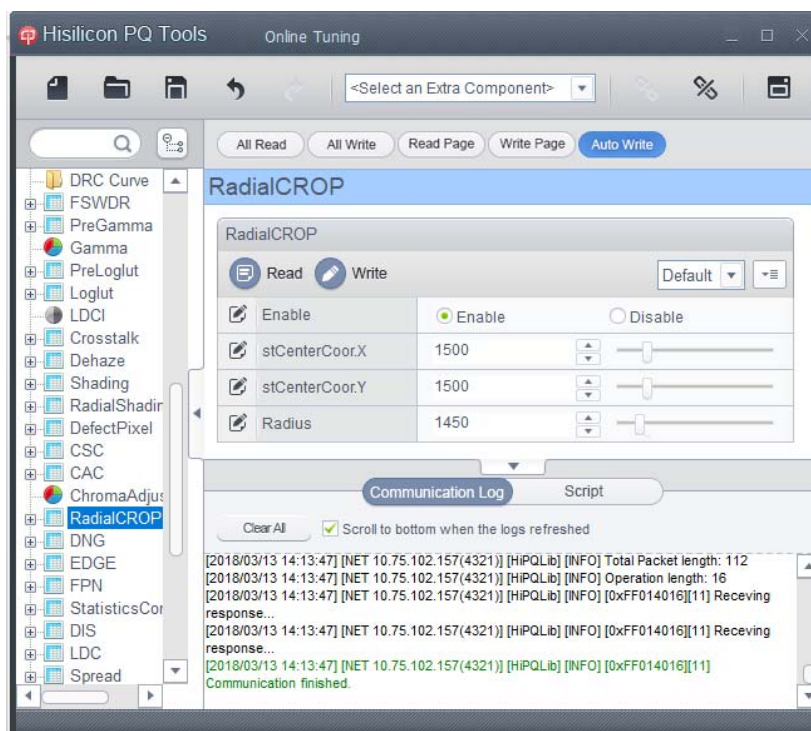
图4-7 鱼眼镜头产线标定前需要配置 RadialCROP



该方案需要在标定数据采集（产线标定图片抓图）前，在 RadialCROP 模块配置鱼眼镜头的圆心和半径，通过半径的缩小，将鱼眼边缘模糊的图像裁掉。



图4-8 Pqtools RadialCROP 的配置

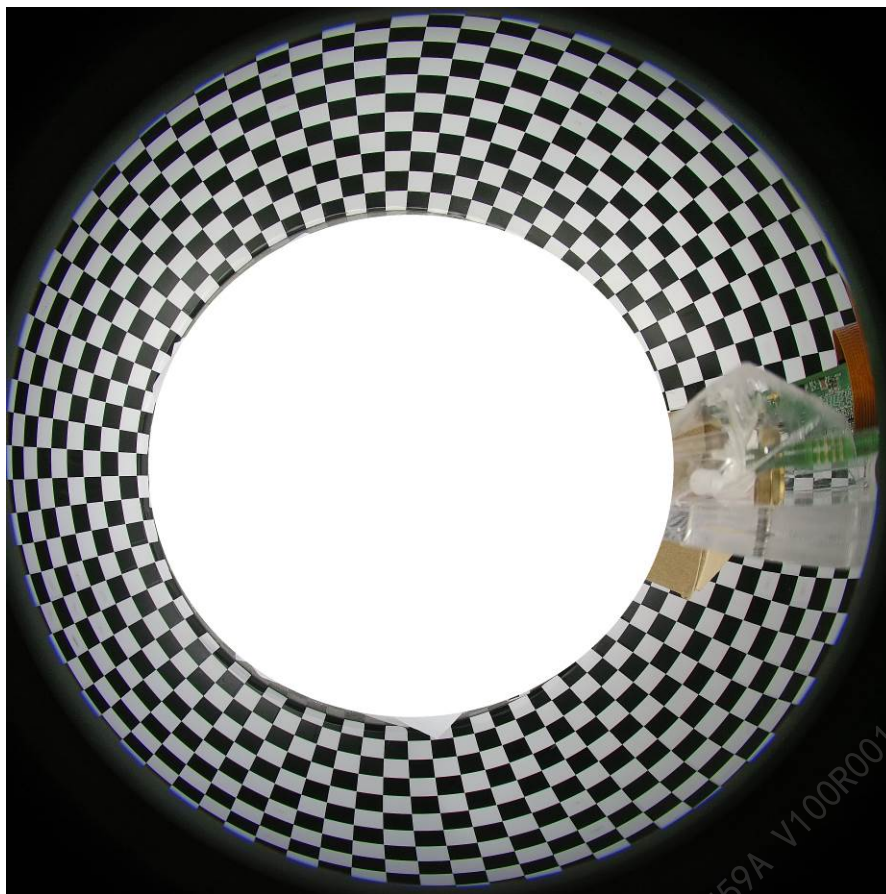


实际在配置时，可以通过预览鱼眼图像，将模糊的边缘裁掉即可，不要裁剪过多的重叠区域。注意使用 PQtools 配置时，需要配好对应的 pipe 编号，多个 pipe 要分别配置一次。

一般来说，一批设备的图像中心、半径 配置一次就可以。中心设定为镜头图像实际的中心，半径的设定可以根据预览鱼眼图像的效果来判断，以可以裁掉鱼眼图像边缘的模糊部分的最小半径即可，注意半径不宜设置过小，以避免损失过多的重叠区。



图4-9 边缘模糊，需要配置 RadialCROP 的图像



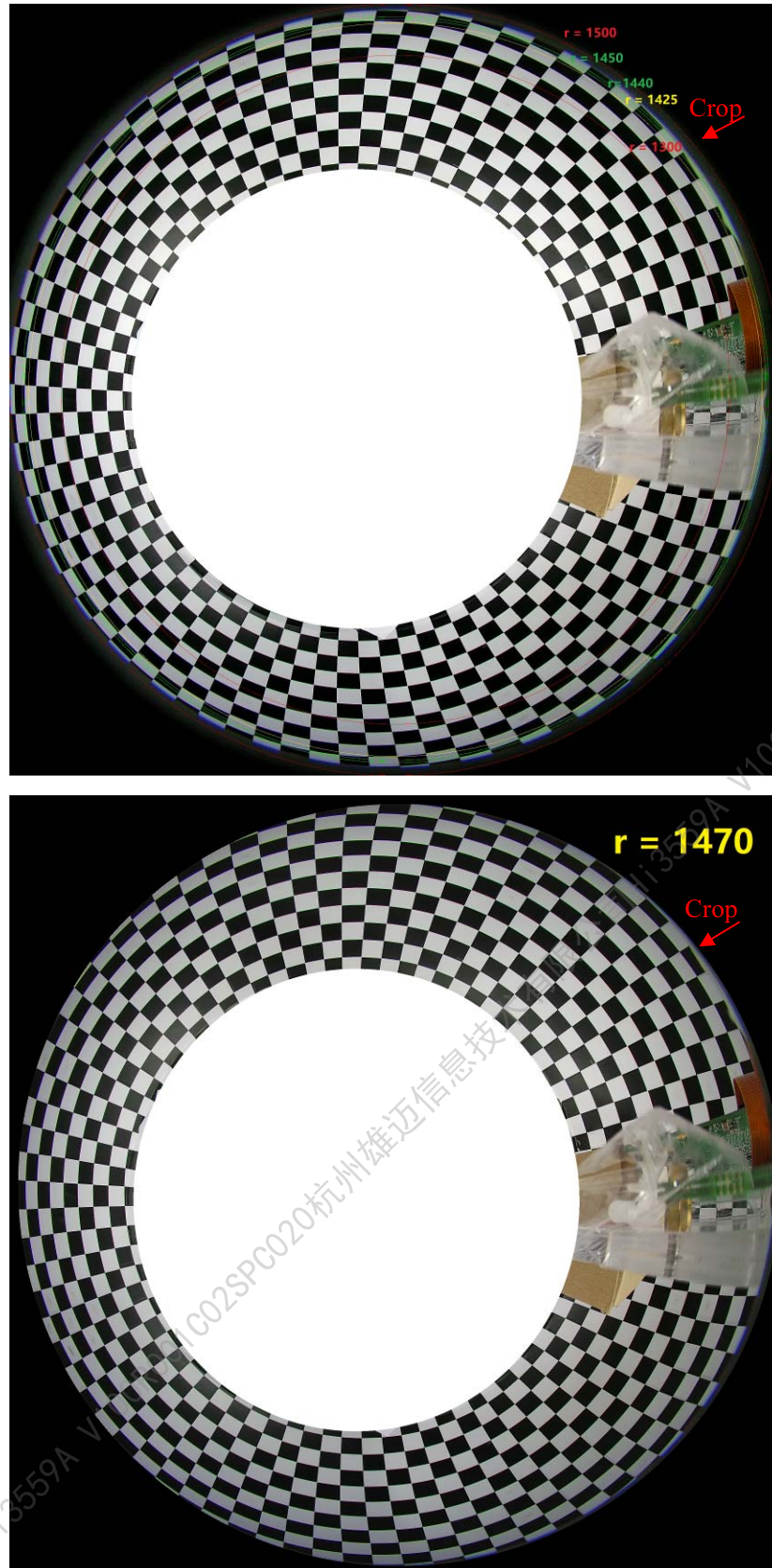
该方案仅在鱼镜头标定时配合使用，非鱼眼的标定不需要配置 RadialCROP；板端图像拼接时可以用不用配置 RadialCROP。

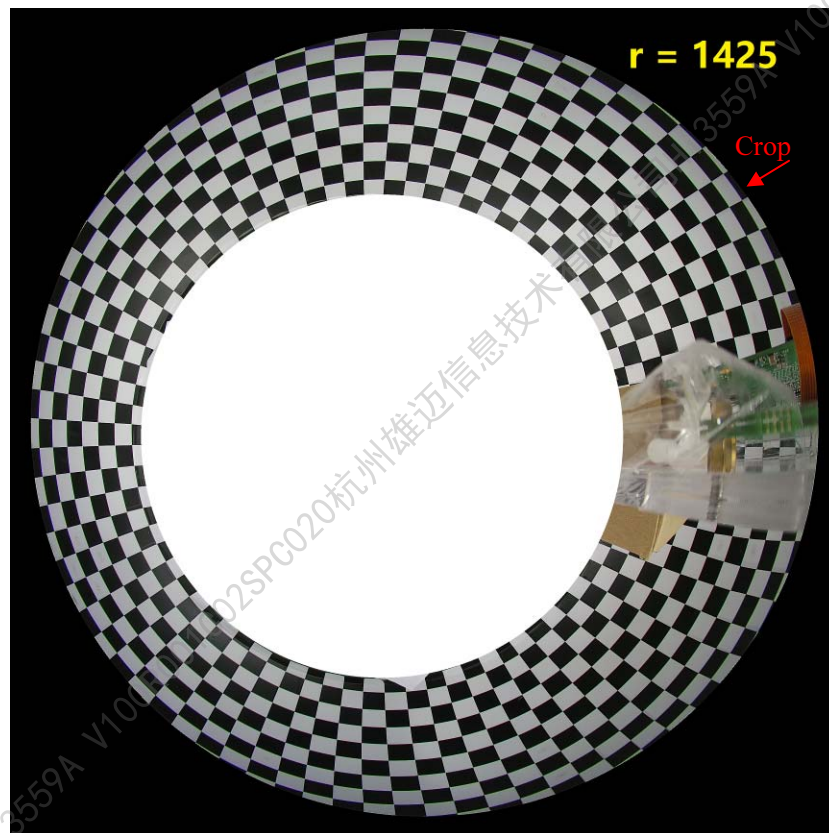
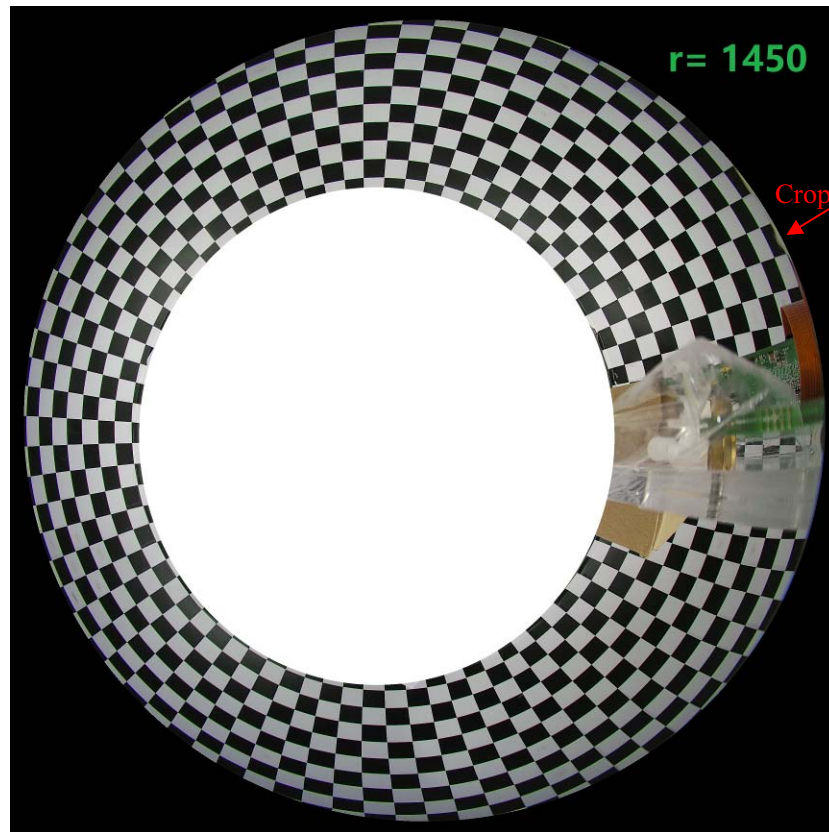
RadialCROP 建议配置方式：

- 使用实时预览，可以同时显示多路鱼眼画面；
- 场景可以选择任意一个实际场景，调整镜头，将有细节的部分移至画面的边缘（以方便观察模糊情况及图像裁剪情况）；
- 配置 RadialCROP，设置图像的圆心位置（根据实际调整），减小半径大小，使得鱼眼的周边全部模糊的边缘可以被完全裁掉，并且要相邻鱼眼图像的边缘要保留足够的重叠区；（参考：按半径的长度来计量，可以裁剪掉重叠区域的 1/4，如重叠的半径范围为[1300,1500]，则可以裁剪半径至 1450）
- 一批产品的 RadialCROP 圆心和半径可以配置相同的参数，不需要针对每台设备单独调整；



图4-10 不同半径的 RadialCROP 效果预览







产线标定主要输出

- 产线标定和模型标定类似，会输出一个更精细且体现设备个体差异的.cal 标定文件。
- 该标定文件也需要进行转换才能直接在板端加载使用。

标定结果的分析

主要通过标定接口的返回值判断结果的优劣，如表 4-3 所示。

表4-3 产线标定结果的判断表

成员名称	描述	参考值
dMaxReprojErr	值越小，标定结果越好	≤ 30
dAverageReprojErr	值越小，标定结果越好	≤ 4
dTotalMatchedPoints	指匹配成功的角点对，与图像数量及棋盘格内角点数相关。	图片的总数*每张图片中角点的个数，若低于该值，说明有图片未成功检测出角点

根据标定结果生成 LUT 及 BBox 表

详细步骤参考 4.3.4 “LUT 及 BBox 表生成”。

以下是标定结果转换的差别：

表4-4 BboxLUT 的差异

项目	对应接口	备注
BBox 表	hiAVS_PolyFromCalibration	仅 Hi3559AV100ES 需要额外调用生成 Bbox 表。
LUT	hiAVS_LutFromCalibration	所有标定结果都需要转换为 LUT 表才能应用。

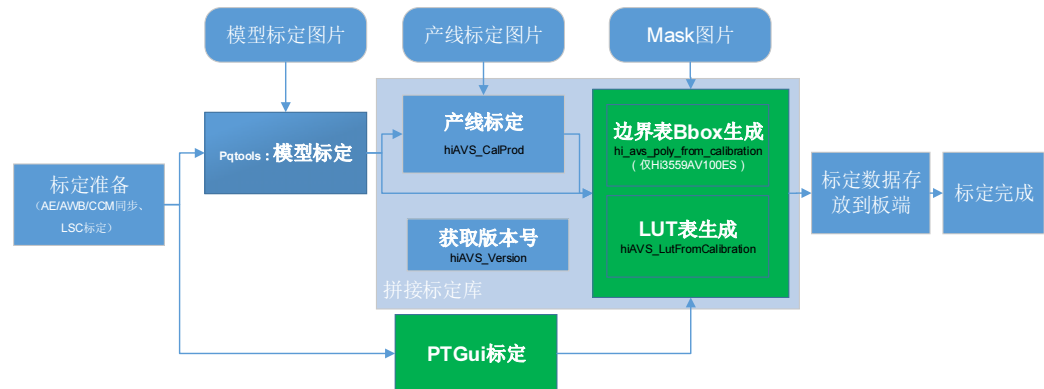
4.3.3 第三方（PTGui、hugin）标定

产线标定库支持第三方标定工具及软件。当前支持的第三方标定工具有：PTGui 和 Hugin。

PTGui 为第三方的全景标定及拼接软件，标定步骤简单易用，为业界通用工具。和 PTGui 类似，Hugin 也是一个常见的拼接标定软件，并且跨平台开源全景摄影图像拼接软件，用户可以将其集成到自己的平台上进行开发应用。第三方标定都可以标定出镜头的内参及外参，它的标定结果需要导入到 AVS 标定库，转换成 AVS 模型所需的查找表。



图4-11 第三方标定在标定流程中所处的环节



注意

- 采用第三方标定不支持修改最佳拼接距离，故生成 LUT 表时最佳拼接距离配置无效，其拼接效果与标定时保持一致。
- PTGui 是商业软件，用户使用需要自行购买 license。

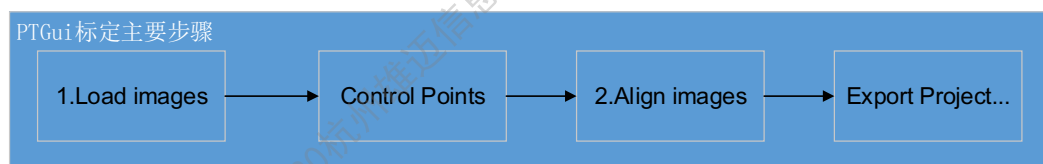
4.3.3.2 第三方标定的数据来源

第三方标定需要抓取 1 组细节纹理丰富的场景的图片，由于只有一组图片，用户需要在最常应用的距离下抓取图片。

4.3.3.3 PTGui 标定的主要步骤

PTGui 具体使用方法可参考其 Help 功能，这里仅简单说明 PTGui 标定的主要操作过程：

图4-12 PTGui 标定的主要流程

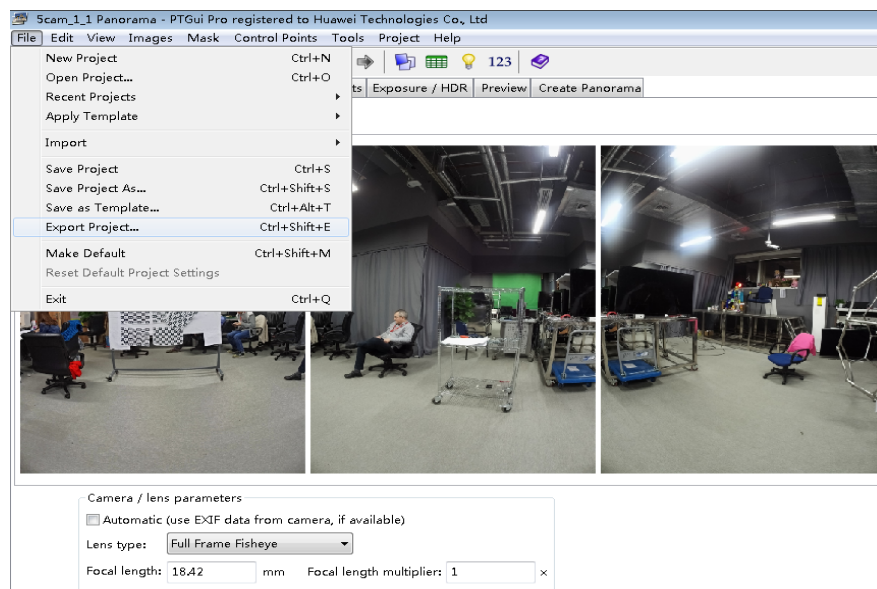


- 在 Project Assistant 中使用 1.Load images... 导入需要标定的图片；
- 在 Control Points 中通过 Auto 或者 Manual 成对标记重叠区域的 Control Points；
- 在 Project Assistant 中使用 2.Align images 预览拼接的区域，初步确认拼接效果；如果效果达不到预期，需要返回步骤 2 重新调整 Control Points；
- 确认好效果后，使用 File / Export Project... (CTRL+ALT+T) 导出模板文件；
- 将 PTGui 导出的文件后缀名修改为 .pto。



标定时需同时抓拍每一路图像，尽量选择细节丰富的场景，有利于特征点检测。PTGui 标定完成后导出标定结果，可在 PTGui 中选择 File/Export Project 打开导出路径，将标定文件后缀修改为.pto，才能在 AVS 标定工具识别。

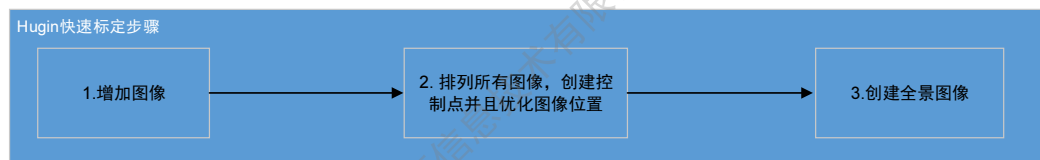
图4-13 导出 PTGui 标定结果



4.3.3.4 hugin标定的主要步骤

本文以 Hugin - Panorama editor 为例介绍其主要操作过程。

图4-14 使用 hugin 快速标定的步骤



步骤 1. 增加图像：同时打开所有需要拼接标定的一组图片，并配置镜头信息。

- 多路鱼眼请使用“圆形鱼眼”；
- 使用非鱼眼，请使用“标准的（直线的）”；

步骤 2. 排列所有图像，创建控制点并且优化图像位置：自动创建控制点并排列图像，只有完成这一步，才可以预览拼接效果，并进行下一步的全景图像创建。

步骤 3. 创建全景图像：生成拼接后的图像，其中 *.pto 存放拼接的标定信息，作为产线标定库第三方标定的结果。

如果以上自动标定结果不理想，需要手动调整控制点，请参考官方文档进行。

---结束



4.3.3.5 第三方标定数据输出

标定文件需要调用 AVS 标定库，参考 4.3.4 “LUT 及 BBox 表生成”中的步骤，将.pto 文件及 Mask 图像转换成系统所需的 LUT（及 BBox 表：仅 Hi3559AV100ES 需要）。



注意

- 采用第三方标定不支持修改最佳拼接距离，故生成 LUT 表时最佳拼接距离配置无效，其拼接效果与标定时保持一致。
- PTGui 是商业软件，用户使用需要自行购买 license。

4.3.3.6 根据标定结果生成 LUT 及 BBox 表

详细步骤 参考 4.3.4 “LUT 及 BBox 表生成”。

以下是标定结果转换的差别：

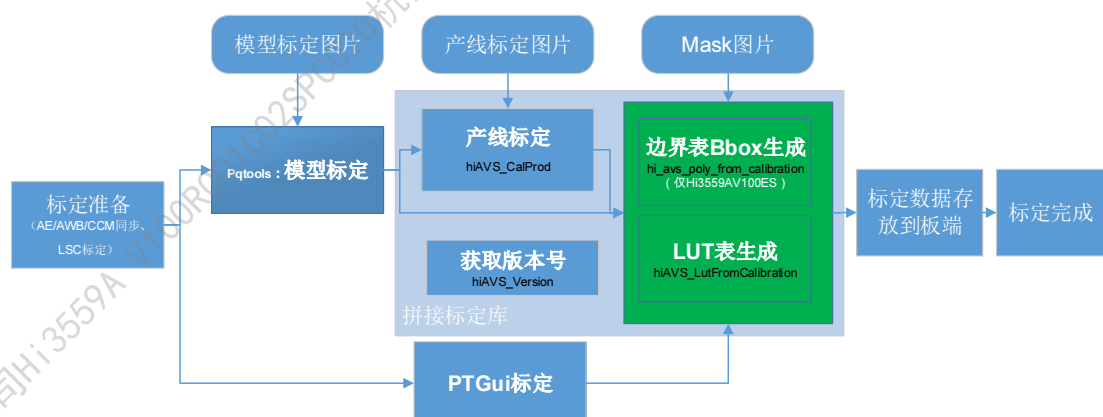
表4-5 Bbox 和 LUT 的差别

项目	对应接口	备注
BBox 表	hiAVS_PolyFromCalibration	仅 Hi3559AV100ES 需要额外调用生成 Bbox 表
LUT	hiAVS_LutFromCalibration	所有标定结果都需要转换为 LUT 表才能应用

4.3.4 LUT 及 BBox 表生成

在模型标定、产品标定或第三方标定及 Mask 图像的基础上，可使用标定工具，配置最佳拼接距离，生成硬件所需的 LUT 查找表及 BBox 有效区域边界表。

图4-15 LUT/Bbox 表生成在标定流程中所处的环节





注意

Hi3559AV100ES 版本需要 LUT 查找表及 BBox 有效区域边界表，其他量产版本仅需要 LUT 查找表。

4.3.4.2 LUT/BBox 表数据来源说明

模型标定生成的.cal 文件或者产线标定生成的.cal 文件或第三方标定生成的标定文件.pto、mask 文件。

4.3.4.3 Mask 文件生成

LUT 表的生成需要制作每路输入的 Mask 模板图像，用于标识出输入图像的有效区域。Mask 图像必须是与输入分辨率相同的.png 图像，白色区域表示有效区域，黑色区域表示无效区域。

- 对于双鱼眼结构，mask 为白色的圆形图像，如图 4-16 所示。
- 对于水平环绕四鱼眼结构，由于水平方向重叠区较大，mask 推荐为椭圆形图像，将水平方向裁剪剩余大约 120° 视角，垂直方向不裁剪，如图 4-16(b)所示。
- 对于非鱼眼图像，一般来说，mask 为全白的图像即可，如果需要裁剪一些重叠区域，则可以制作成白色的矩形图像，如图 4-16(c)和(d)所示。
- 大部分情况下，并不需要裁剪掉部分重叠区域，只需要确保重叠区域不低于 20% 即可，当系统处理压力较大时，可以通过调整 Mask 的有效区域，减小重叠区，从而减小系统的处理性能压力。但需要特别注意的是，重叠区的裁剪必须格外小心，裁剪过多会导致融合区不够平滑，影响拼接效果。
- 一般情况下，有效区域的边缘定义不需要太精确，在圆饼图半径及中心点个体差异小于 2% 时，相同产品类型的相机可以共用一套 mask 图片。若个体差异大于 2%，则可能造成融合区清晰度下降或者融合区大小不对称问题。而非鱼眼图像不存在无效区域，故一般情况下相同产品类型的相机共用一套 mask 图片即可。

图4-16 Mask 图像示例



(a) 双鱼眼 mask 示例 (b) 四鱼眼 mask 示例 (c) 非鱼眼 mask 示意图 1 (d) 非鱼眼 mask 示意图 2

mask 文件可以使用绘图软件通过可视化步骤简单绘制，以绘图工具 Photoshop 为例，介绍 mask 图像的生成：



为快速生成 mask 图像，直接使用每个 Camera 抓取的图像（摄像头所在的场景可任意），参考该图像绘制 mask 图像。

针对鱼眼镜头：

- 使用绘图工具 Photoshop 打开 从 Camera 中抓出来的图像；
- 使用椭圆工具，使得该形状正好覆盖正常图像显示区域；
- 将该区域填充为纯白色（RGB=0x000000），边框填充为纯黑或纯白或无边框；
- 使用矩形工具选择整幅图像，填充为纯黑色（RGB=0xFFFFFFFF），边框填充为纯黑或无边框，将该矩形设置为最底层；
- 另存为 Camera_mask_X.png 文件；（X 为镜头编号）
- 其他鱼眼镜头也采用相同的方式。

针对普通镜头：

- 使用绘图工具 Photoshop 打开 从 Camera 中抓出来的图像；
- 使用矩形工具选择需要覆盖的有效图像部分（需要包含重叠区域），填充为纯白色（RGB=0x000000），边框填充为纯白或无边框；
- 使用矩形工具选择整幅图像，填充为纯黑色（RGB=0xFFFFFFFF），边框填充为纯黑或无边框，将该矩形设置为最底层；
- 另存为 Camera_mask_X.png 文件；（X 为镜头编号）

Photoshop 也可以更换为其它可以对图层操作的图像处理工具，方法类似。



注意

Adobe Photoshop 是商业化软件，不在发布包范围内，需要用户自行购买；当然有其他绘图软件也可以完成该工作。

4.3.4.4 使用 PQtools 完成 LUT 及 BBox 表的生成

详细的 LUT/Bbox 的生成可以参考文档《拼接调试指南》。



5 API 应用实例

5.1 标定库应用流程

使用流程请参考图 4-3。

5.2 一个完整的 Sample 示例

```
#include <iostream>
#include <windows.h>
#include <ctime>
#include "hi_type.h"
#include "hi_avs_prod_calib.h"
HI_S32 main(int argc, char **argv)
{
    HI_S32 s32Ret = 0;
    DWORD start_time = GetTickCount();
    DWORD end_time = GetTickCount();
    printf_s("\n\n");

    /*1.hiAVS_Version*/
    HI_CHAR cAVSCalibVersion[128];
    hiAVS_Version(cAVSCalibVersion);
    printf_s("\t1.Welcome to use hisilicon AVS Calibration Library %s . ",
cAVSCalibVersion);
    printf_s("\n\n");

    /*2.hiAVS_CalProd*/
    printf_s("*****2.hiAVS_CalProd*****\n");
    HI_U32 camera = 2;
    HI_CHAR input_cal_file[MAX_PATH] =
"..\\..\\test\\for_prod_cal\\prod_step1.cal";
```




```
HI_FLOAT fixture_radius = (HI_FLOAT)0.65;
const HI_CHAR * pcCalImage[2];
pcCalImage[0] = "..\\..\\test\\for_prod_cal\\camera_0.JPG";
pcCalImage[1] = "..\\..\\test\\for_prod_cal\\camera_1.JPG";

HI_CHAR output_cal_file[MAX_PATH] =
"..\\..\\test\\for_prod_cal\\prod_step2_V4.cal";
HI_AVS_CAL_STITCH_MEASUREMENT_S stResult;

start_time = GetTickCount();
s32Ret = hiAVS_CalProd(
    camera,
    input_cal_file,
    output_cal_file,
    fixture_radius,
    pcCalImage,
    stResult);
end_time = GetTickCount();

if (s32Ret != HI_SUCCESS)
{
    printf_s("!!Error: __%d__, ret=%d \n", __LINE__, s32Ret);
    printf_s("\t\tcamera: %u \n", camera);
    printf_s("\t\tinput_cal_file: %s \n", input_cal_file);
    printf_s("\t\toutput_cal_file: %s \n", output_cal_file);
    printf_s("\t\tfixture_radius: %f \n", fixture_radius);
    printf_s("\t\tpcCalImage[0]: %s \n", pcCalImage[0]);
    printf_s("\t\tpcCalImage[1]: %s \n", pcCalImage[1]);
}
else
{
    printf_s("\thiAVS_CalProd:sucess ,time=%ld ms \n", end_time -
start_time);
    printf_s("\t\tmaximum_reproj_err: %f \n", stResult.dMaxReprojErr);
    printf_s("\t\taverage_reproj_err: %f \n",
stResult.dAverageReprojErr);
    printf_s("\t\ttotal_matched_points: %f \n",
stResult.dTotalMatchedPoints);
}
printf_s("\n\n");

/*3.hiAVS_LutFromCalibration*/
printf_s("*****3.hiAVS_LutFromCalibration*****\n");
HI_CHAR cCalibrationFile[] =
```




```
    "..\\..\\test\\for_mesh_poly\\2fisheye.cal";
    HI_CHAR cMaskPrefix[] = "..\\..\\test\\for_mesh_poly\\2fisheye_mask";
    HI_FLOAT fStitchDistance = 4;
    HI_AVS_LUT_ACCURACY_E enLutAccuracy = HI_AVS_LUT_ACCURACY_HIGH;
    HI_BOOL bIsHiAvsCal = HI_TRUE;

    HI_CHAR cOutputPrefix[] = "..\\..\\test\\for_mesh_poly\\2fisheye_lut";

    s32Ret = hiAVS_LutFromCalibration(cCalibrationFile, cMaskPrefix,
    cOutputPrefix, fStitchDistance, enLutAccuracy, bIsHiAvsCal);

    if (s32Ret != HI_SUCCESS)
    {
        printf_s("!!Error: __d__, ret= %d \n", __LINE__, s32Ret);
        printf_s("\\t\\tcCalibrationFile: %s\\n", cCalibrationFile);
        printf_s("\\t\\tcMaskPrefix: %s\\n", cMaskPrefix);
        printf_s("\\t\\tcOutputPrefix: %s\\n", cOutputPrefix);
        printf_s("\\t\\tfStitchDistance: %f \n", fStitchDistance);
        printf_s("\\t\\tfenLutAccuracy: %d\\n", enLutAccuracy);
        printf_s("\\t\\tbIsHiAvsCal: %d \n", bIsHiAvsCal);
    }
    else
    {
        printf_s("\\thiAVS_LutFromCalibration:sucess! \n");
    }
    printf_s("\\n\\n");

    /*4.hiAVS_PolyFromCalibration,only used for Hi3559AV100ES*/
    printf_s("*****4.hiAVS_PolyFromCalibration*****\\n");

    start_time = GetTickCount();
    s32Ret = hiAVS_PolyFromCalibration(cCalibrationFile, cMaskPrefix,
    cOutputPrefix, fStitchDistance, bIsHiAvsCal);
    if (s32Ret != HI_SUCCESS)
    {
        printf_s("!!Error: __d__, ret= %d \n", __LINE__, s32Ret);
        printf_s("\\t\\tcCalibrationFile: %s \n", cCalibrationFile);
        printf_s("\\t\\tcMaskPrefix: %s \n", cMaskPrefix);
        printf_s("\\t\\tcOutputPrefix: %s \n", cOutputPrefix);
        printf_s("\\t\\tfStitchDistance:%f \n", fStitchDistance);
        printf_s("\\t\\tbIsHiAvsCal: %d\\n", bIsHiAvsCal);
    }
    else
    {
```



```
printf_s("\\thiAVS_PolyFromCalibration:suceess!  \n");  
}  
printf_s("\\n\\n");  
return HI_SUCCESS;  
}
```

5.3 Sample 编译运行的依赖条件简述

Sample 示例要实际运行起来，以下依赖条件需要根据实际情况配置。

No	依赖条件	获取方式	备注
1	工程配置	将版本发布包中的标定库链接到开发工程中，详细过程可参考 4.2.2.2 “使用 VS2015 创建开发环境” 。	
2	模型标定文件	通过 PQtools，生成模型标定文件*.cal，模型标定的输出文件作为产线标定的输入。	
3	产线标定图片	在产线标定环境中抓取标定图片，注意对于鱼镜头头还需要抓取前配置好 RadialCROP，过程和要求参考章节 4.3.2.2 “产线标定” 。	
4	产线标定环境半径	需要实际测量标定环境的半径（也即测试设备的镜头距离标定图卡的距离）。	
5	Mask 文件	根据多路原始输入图像，制作好 mask 文件，制作过程可参考 4.3.4.3 “Mask 文件生成” 。	
6	OpenCV 动态库	编译时可以不需要 OpenCV 动态库，但运行时必需 OpenCV 动态库，OpenCV 的获取及生成请参考 4.2.1 “下载和使用 OpenCV” 。	