

Rockchip 双目拼接产线标定指南

发布版本：V1.1.1

日期：2023.06.20

文件密级：绝密 秘密 内部资料 公开

免责声明

本文档按“现状”提供，瑞芯微电子股份有限公司（“本公司”，下同）不对本文档的任何陈述、信息和内容的准确性、可靠性、完整性、适销性、特定目的性和非侵权性提供任何明示或暗示的声明或保证。本文档仅作为使用指导的参考。

由于产品版本升级或其他原因，本文档将可能在未经任何通知的情况下，不定期进行更新或修改。

商标声明

“Rockchip”、“瑞芯微”、“瑞芯”均为本公司的注册商标，归本公司所有。

本文档可能提及的其他所有注册商标或商标，由其各自拥有者所有。

版权所有 © 2021 瑞芯微电子股份有限公司

超越合理使用范畴，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。

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址：福建省福州市铜盘路软件园A区18号

网址：www.rock-chips.com

客户服务电话：+86-4007-700-590

客户服务传真：+86-591-83951833

客户服务邮箱：fae@rock-chips.com

前言

本文为应用双目拼接方案进行开发的程序员而写，目的是介绍双目拼接产线标定基本原理、操作步骤及使用注意事项等内容。

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
RV1106	
RV1126	

修订记录

日期	版本	作者	修改说明
2023.03.07	V0.1.1	mingpei.liu	初始版本
2023.03.15	V0.1.2	mingpei.liu	完善场景布局说明
2023.04.18	V0.1.3	mingpei.liu	添加标定异常分析
2023.06.06	v0.1.4	mingpei.liu	完善标定异常分析
2023.06.20	v1.1.1	mingpei.liu	引入模型标定文件引导标定
2023.07.10	v1.1.2	mingpei.liu	完善拍图注意事项

1. 概述

1.1 双目拼接概述

双目拼接可以将两路输入图像拼接为一路图像，从而实现获取更大视野的需求。

1.2 产线标定概述

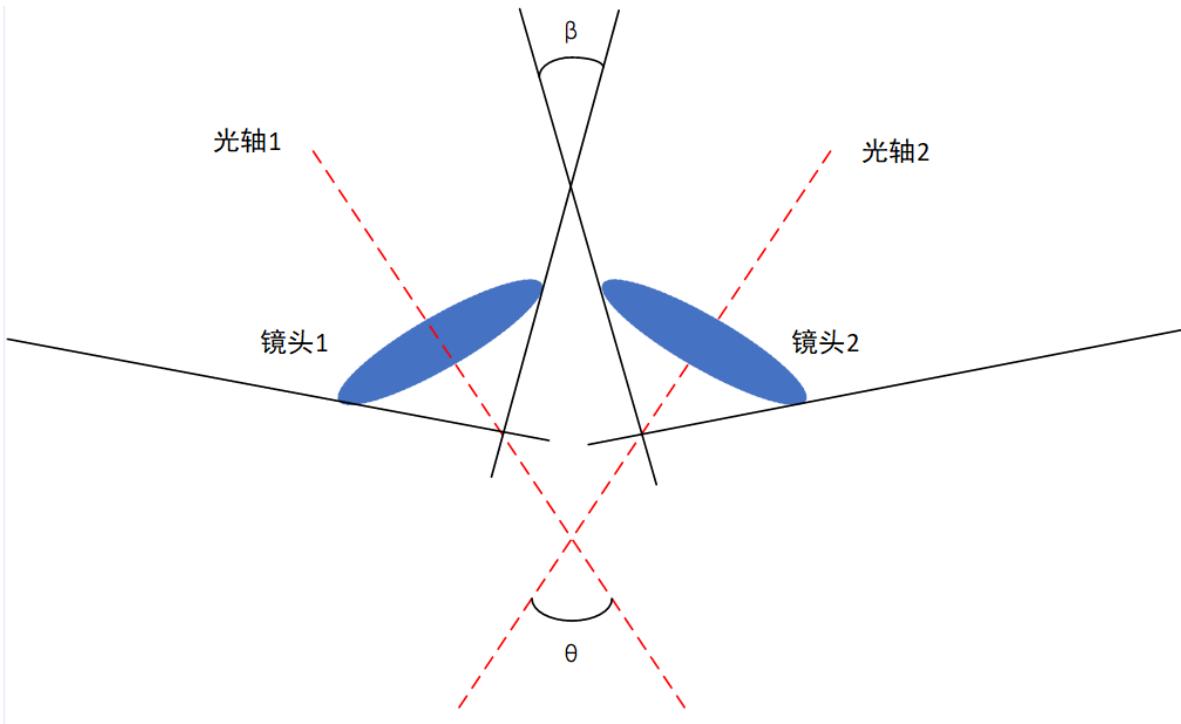
双目拼接产线标定通过对所提供棋盘格模具的单张拍摄，借助模型标定的标定结果作为种子文件来获得立体镜头模组的所有标定参数。标定参数包括左右镜头各自的内参、左右相机的相对外参。内参包括焦距 (f_x, f_y)，主点位置 (u_0, v_0) 以及光学失真系数 (k_1, k_2, k_3, p_1, p_2)；相对外参包括三轴角度 $Tilt, Pan, Roll$ ， x 轴 y 轴 z 轴的平移参数；同时返回所得标定参数的相对误差。

2. 模组设计要求

模组设计要从需求出发，根据需求对sensor和镜头进行选型，并设计合适的光轴夹角。

2.1 模组设计指标

图2.1 模组设计示意图



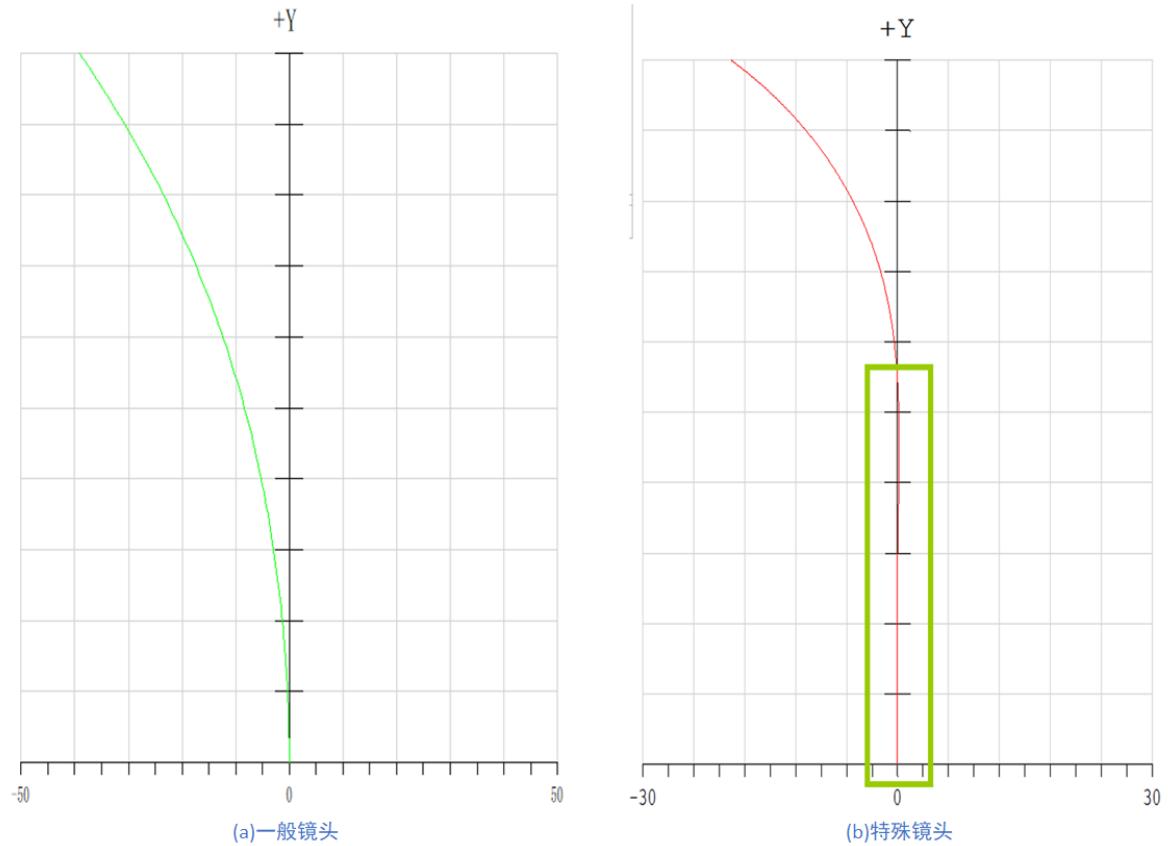
如图2.1所示为双目模组设计示意图，具体设计要求如下：

1. 两个相机光轴之间水平夹角为 θ ；
2. 两个相机光轴之间垂直夹角为 0° ；
3. 两个相机水平方向重叠区域的FOV为 β ($\beta > 15^\circ$)；
4. 要求工厂组装时组装精度为 $\pm 3^\circ$ ，既水平夹角为 $\theta \pm 3^\circ$ ，垂直夹角为 $\pm 3^\circ$ 。

相机光轴之间的水平夹角 θ 可根据需求进行调整： θ 越大重叠区域越小，拼接输出FOV越大； θ 越小重叠区域越大，拼接输出FOV越小。当光轴之间的水平夹角 θ 固定时：镜头的FOV越大，拼接输出的FOV越大；镜头的FOV越小，拼接输出的FOV越小。

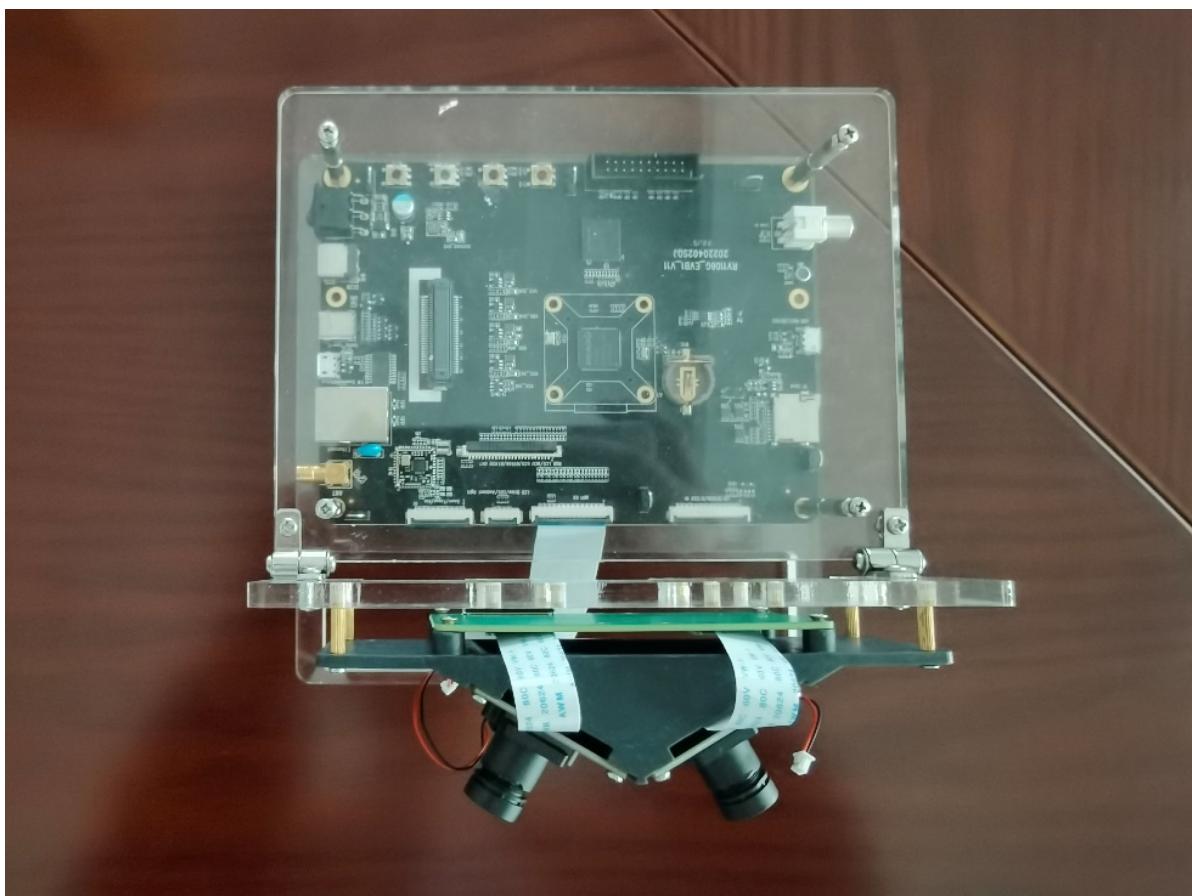
镜头选型上选择一般的广角镜头，不支持经过特殊处理过的镜头，如图2.2所示为镜头的畸变映射曲线，图(a)属于一般镜头的畸变映射曲线，图(b)是经过特殊处理的镜头的畸变映射区域，绿色区域做了无畸变处理。

图2.2 畸变映射曲线图



2.2 模组设计实例

图2.3 RK双目模组实例



如图2.3所示为RK双目模组实例图，其参数信息如下：

- 1) sensor : GC2063
- 2) 光轴水平夹角60°，光轴垂直夹角0°；

- 3) 单目水平FOV109°、垂直FOV59°；
- 4) 拼接水平FOV 170°、拼接垂直FOV53°。

本文的参数设计信息都是以该模组为例进行设计的，包括棋盘格规格设计和标定场景布局。

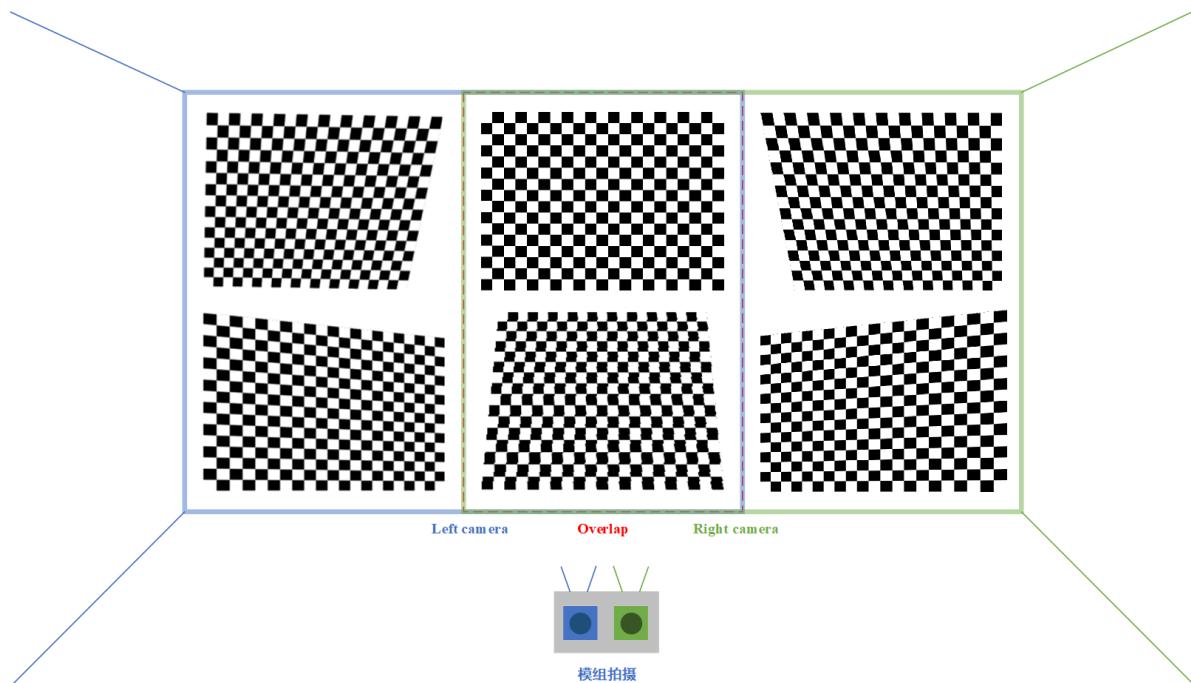
3. 模组标定

双目拼接模组产线标定通过对所提供的棋盘格模具的单张拍摄，来获得立体镜头模组的所有标定参数。

3.1 操作环境

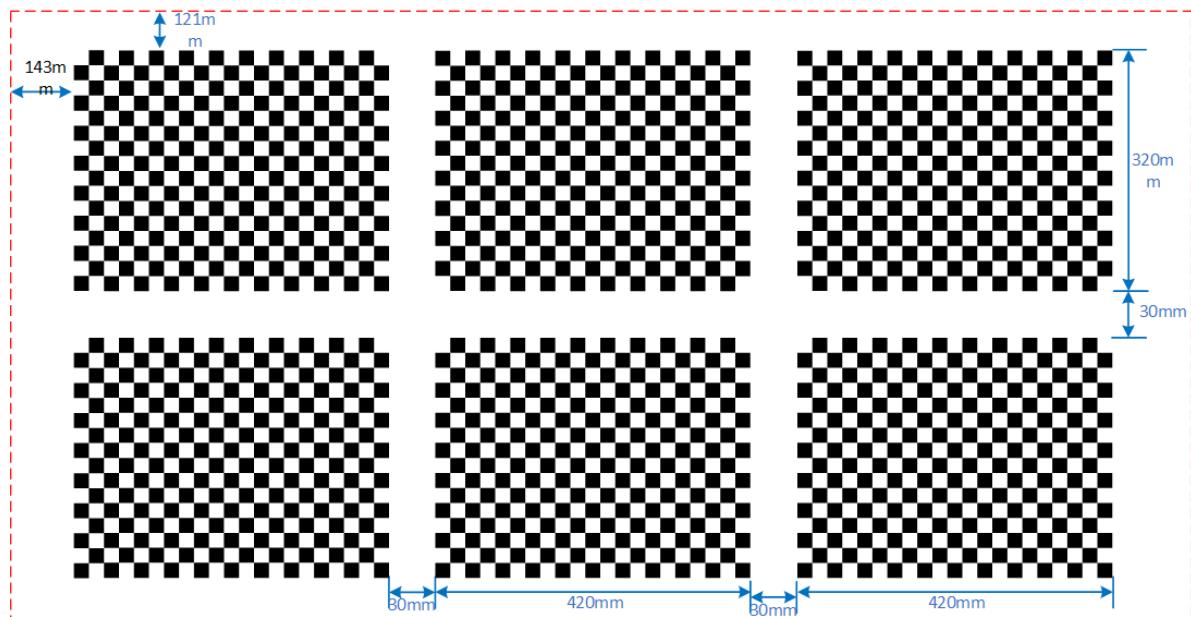
3.1.1 操作环境设计

图3.1 操作环境



如图3.1所示，为模组标定时的场景示意图。每次拍摄标定板时，要求模组的左侧相机能够完整拍摄到左侧的四块棋盘格如图中蓝色框所示，右侧相机能够完整拍摄到右侧的四块棋盘格如图中绿色框所示，标定模具中的六个棋盘格的摆放位置如图3.2所示。

图3.2 未旋转时棋盘格摆放位置



如图3.2所示，为六个棋盘格没有旋转时的摆放位置和棋盘格尺寸。其中外圈的红线表示两相机视野的边界，标定板与标定板之间有大于30mm的距离（可根据实际情况适当调整），每个棋盘格的尺寸为500mm*280mm（该尺寸以2.2中模组为例进行设计）。

图3.3 正式标定时棋盘格摆放位置

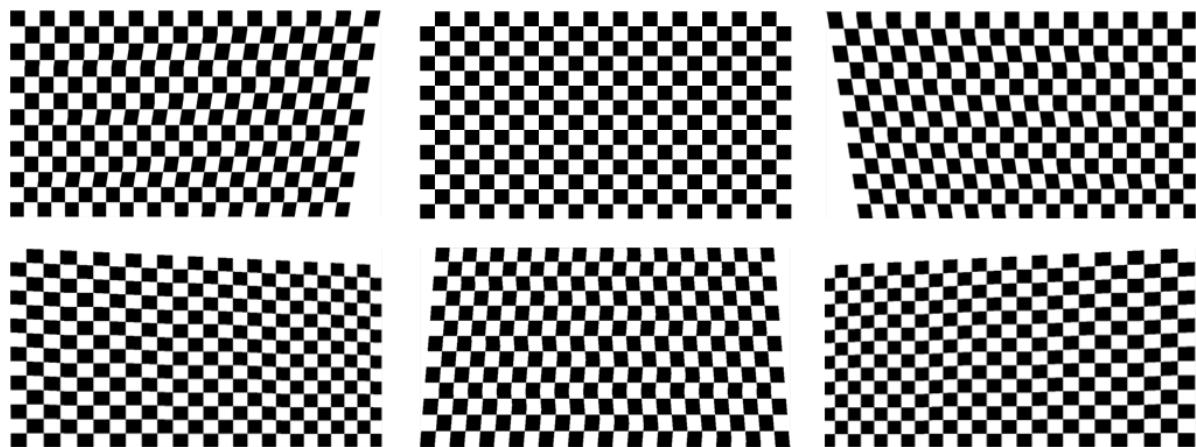
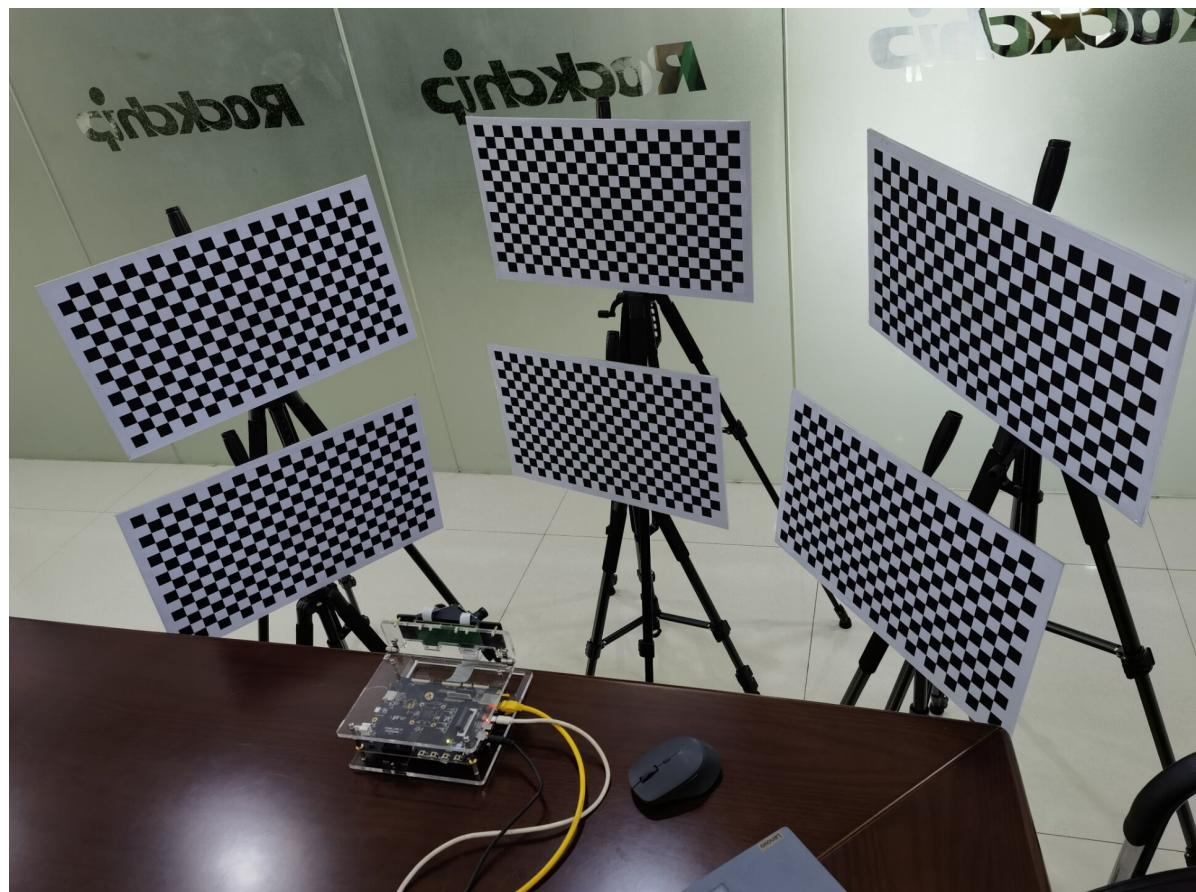


图3.3为正式标定时六个棋盘格的摆放位置。标定过程中需要注意：

1. 任意一个相机所拍摄的**四个棋盘格不能在同一个平面上**，可根据棋盘格在视野中的位置进行适当调整每个棋盘格的旋转角度；
2. 6个棋盘格图像（包括中间留白）的FOV在视野中所占比例应控制在80%到90%之间，**尽量保证棋盘格覆盖较大视野**；
3. 要求**棋盘格必须全部处于两个相机的视野内**，左侧四个棋盘格全部处于左侧相机的视野内，右侧四个棋盘格全部处于右侧相机的视野内（包括棋盘格留白区域）。

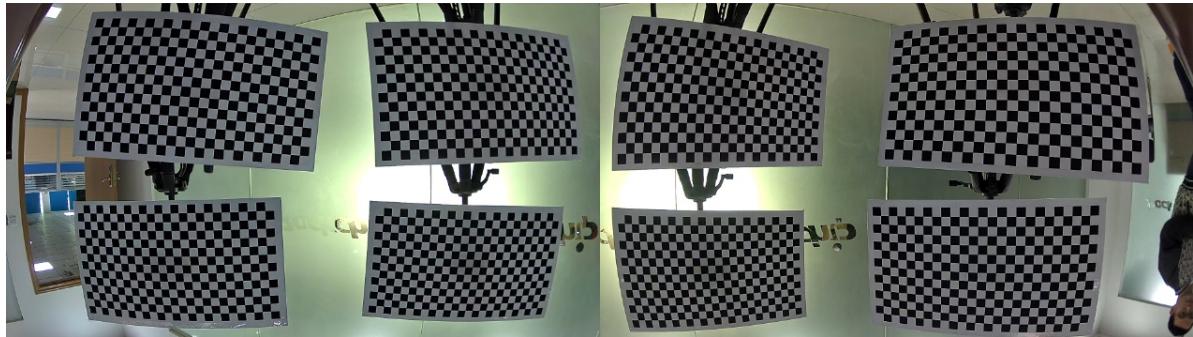
3.1.2 操作环境实例

图3.4 操作环境实例



如图3.4所示为操作环境实例，为了方便调整六个棋盘格所在位置，采用六个三脚架的结构。工厂产线标定时，可先确定拍摄距离和棋盘格摆设位置后根据实际需求设计成固定结构。如3.1.1中所述，棋盘格之间必须有一定间隔，六个棋盘格尽可能占据较大的相机视野。实际拍摄的图像如图3.5所示。

图3.5 拍摄预览图



3.2 标定场景布局定义

在标定场景中，6个不同棋盘格按照3.1所述进行布局。具体拍摄要求如下：

1. 棋盘格距离定义为保证满足3.1所诉覆盖面要求下的距离，同时在该距离下可以清晰的拍摄到棋盘格上的角点。
2. 拍摄时，要求相机曝光可以获取到清晰的棋盘格图像。
3. 要求棋盘格的长边与sensor的长边平行。
4. 棋盘格图像中的反差标准为大于80。亮度灰度在180以上，暗板灰度在100以下（如果白格和黑格亮度没办法始终满足这个条件，则需保证相邻黑白格之间的平均灰度差必须在80以上）。画面内，棋盘格必须保证亮度尽量均匀，不允许出现过亮或者过暗的区域。
5. 环境光强度大于500Lux小于1000Lux，不允许出现过曝现象，需保证拍摄图像的角点清晰可见。
6. 坐标系选择右手坐标系，X轴水平向右，Y轴竖直向下，Z轴垂直向里；
7. 每个旋转都以过棋盘格中心的轴作为旋转轴。
8. 棋盘格所占区域FOV应该达到总视野的80%~90%。
9. 拍摄时背景最好是纯黑或者纯白背景色，背景纹理尽可能少（如背靠白墙拍摄）。
10. 若使用背光板，其超出棋盘格的部分不能超过10mm(最好控制在0mm，即背光板与棋盘格边缘贴合)。
11. **FOV不同，距离会根据需要做调整：**FOV变大，距离适当靠近；FOV变小，距离适当放远一些；对于该模组，由于中间两块棋盘格需要两个相机同时拍摄得到，共同视野的FOV相对小一些，因此中间两个棋盘格可以适当放远一些，两边的四块棋盘格可以适当放近一些。
12. 模组到棋盘格的距离建议设计为可以调节的。

使用背光板的拍摄效果更佳，建议采用背光板拍摄。如果没有背光板，最好保证第9条建议，拍摄背景色尽可能是纯白或者纯黑（如背靠白墙拍摄）。

3.3 棋盘格规格设计

棋盘格规格设计要遵循以下条件：1) 指定标定距离下每个相机可以拍摄到四张完整棋盘格；2) 四张棋盘格尽量覆盖相机视野的80%—90%。

以RK双目拼接模组为例，sensor和镜头信息如下：

- sensor: GC2063，像元尺寸2.8umx2.8um，分辨率1920x1080;
- 焦距: 2.8mm
- 拍摄距离: 600mm
- 镜头水平FOV109°，镜头垂直FOV59°。

通过以上信息可以计算出拍摄距离为600mm处相机视野的尺寸：

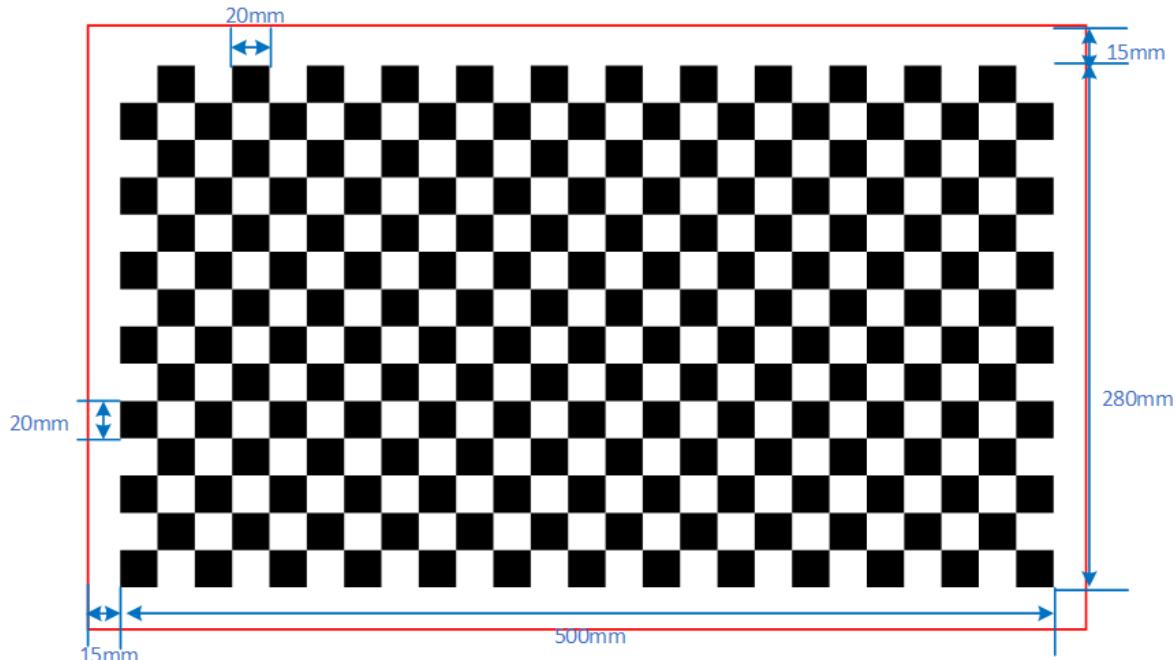
- 水平尺寸: $1920 \times 2.8\text{um} \times 600\text{mm} / 2.8\text{mm} = 1152\text{mm}$ ；

- 垂直尺寸： $1080 \times 2.8 \text{um} \times 600\text{mm} / 2.8\text{mm} = 648\text{mm}$ 。

假设棋盘格占据视野的90%，那么在600mm处拍摄时，棋盘格需要覆盖1037mmx584mm的视野；由于每个相机视野需要四块棋盘格覆盖，同时棋盘格之间至少有30mm的距离，所以每个棋盘格尺寸按504mmx292mm计算；假设每个格子尺寸为20mmx20mm，那么最终棋盘格个数为25x14，最终棋盘格尺寸为500x280。

在棋盘格设计过程中棋盘格图像中格子个数和棋盘格图像中每个格子的尺寸可根据实际需求进行调整，但棋盘格图像的边界处应保留大于15mm的留白区域。

图3.6 棋盘格规格示意图



根据上诉计算模组标定中使用的棋盘格，是对称的方形棋盘格，棋盘格的具体参数如下：

- 棋盘格的长为500mm，宽为280mm；
- 棋盘格中每个方块的物理尺寸是20mm，精度要求为0.1mm；
- 棋盘格要求表面平整；
- 可采用背投发光方式设计棋盘格；
- 棋盘格四周要有留白区域，最好存在大于15mm的留白区域；
- 棋盘格的长边共25个方格，短边共14个方格。

3.4 操作步骤

- 按照规格打印棋盘格，将6个棋盘格按照上面所描述的样子摆放；
- 将双摄摄像头放置在距离棋盘格合适的位置（保证拍摄视野满足设计需求），两个摄像头连线的中心对准6个棋盘格模具的中心，使左右摄像头均能拍摄到4个棋盘格；
- 双摄摄像头同步对焦至拍摄距离处，使得棋盘格显示清晰，同时拍摄，获取左右图片；
- 将左图、右图，连同配置文件一齐作为输入，获得立体标定的所有参数。

3.5 API接口及Demo

3.5.1 接口定义

```
/*
 * @enum      rkAVS_CALIB_STATUS_E
 * @brief     the return status of function
 */
typedef enum rkAVS_CALIB_STATUS_E
```

```

{
    RK_AVSCALIB_STATUS_EOF = -1,           /**< error of function inside */
    RK_AVSCALIB_STATUS_OK = 0,             /**< run successfully */
    RK_AVSCALIB_STATUS_FILE_READ_ERROR,   /**< error: fail to read file */
    RK_AVSCALIB_STATUS_FILE_WRITE_ERROR,  /**< error: fail to write file */
    RK_AVSCALIB_STATUS_INVALID_PARAM,    /**< error: invalid parameter */
    RK_AVSCALIB_STATUS_ALLOC_FAILED,     /**< error: fail to alloc buffer
*/
    RK_AVSCALIB_STATUS_BUTT            /**< reserved fields */
} AVSCALIB_STATUS_E;

```

rkAVS_CALIB_STATUS_E定义了算法库内部的状态信息。

```

/*
 * @enum      rkAVS_CALIB_IMAGE_FORMAT_E
 * @brief     Specify the image format.
 * @note      Have gray, rgb, yuvnv12.
 */
typedef enum rkAVS_CALIB_IMAGE_FORMAT_E {
    RK_AVSCALIB_IMAGE_FORMAT_TYPE_GRAY = 0,          /**<gray
format*/
    RK_AVSCALIB_IMAGE_FORMAT_TYPE_RGB = 1,            /**<rgb format*/
    RK_AVSCALIB_IMAGE_FORMAT_TYPE_YUVNV12 = 2,        /**<yuvnv12
format*/
} AVSCALIB_IMAGE_FORMAT_E;

```

rkAVS_CALIB_IMAGE_FORMAT_E定义了输入图像的基本格式，包括灰度图、RGB图和YUVNV12图。

```

/*
 * @enum      rkAVS_CALIB_IMAGE_PARAMS_S
 * @brief     calibration image params.
 */
typedef struct rkAVS_CALIB_IMAGE_PARAMS_S
{
    uint32_t u32ImageWidth;                      /**<image
width*/
    uint32_t u32ImageHeight;                     /**<image
height*/
    uint32_t u32ImageStride;                    /**<image
stride*/
    AVSCALIB_IMAGE_FORMAT_E eImageFormat;       /**<image format*/
} AVSCALIB_IMAGE_PARAMS_S;

```

rkAVS_CALIB_IMAGE_PARAMS_S定义了输入图像的基本信息，保留图像的宽、高、stride和图像格式。

```

/*
 * @enum      rkAVS_CALIB_IMAGE_PARAMS_S
 * @brief     calibration image params.
 */
typedef struct rkAVS_CALIB_IMAGE_DATA_S
{
    AVSCALIB_IMAGE_PARAMS_S stImageParams;
    uint8_t* pu8ImageData;
} AVSCALIB_IMAGE_DATA_S;

```

rkAVS_CALIB_IMAGE_DATA_S 定义了输入图像的参数信息。

```
/*
 * @enum      rkAVS_CALIB_BOARD_PARAMS_S
 * @brief     chess board params.
 */
typedef struct rkAVS_CALIB_BOARD_PARAMS_S
{
    uint32_t u32BoardWidth;
    /**<coners number of board width*/
    uint32_t u32BoardHeight;                                         /**
<coners number of board height*/
    uint32_t u32SquareSize;
    /**<square size*/
} AVS_CALIB_BOARD_PARAMS_S;
```

rkAVS_CALIB_BOARD_PARAMS_S 定义了标定板的基本信息，包括标定板水平方向角点的个数 u32BoardWidth、标定板垂直方向角点的个数 u32BoardHeight 和棋盘格每个角点的尺寸(mm)。

```
/*
 * @enum      rkAVS_CALIB_CAMERA_TYPE_E
 * @brief     Specify the camera type.
 * @note      Have pin hole, CMei, fisheye.
 */
typedef enum rkAVS_CALIB_CAMERA_TYPE_E {
    AVS_CAMERA_TYPE_PINHOLE = 0,                                     /**
< PIN HOLE*/
    AVS_CAMERA_TYPE_OMN = 1,                                         /**
< CMei*/
    AVS_CAMERA_TYPE_FISHEYE = 2,                                       /**
<fisheye*/
} AVS_CALIB_CAMERA_TYPE_E;
```

rkAVS_CALIB_CAMERA_TYPE_E 定义了相机标定的模型。

```
/*
 * @enum      rkAVS_CALIB_CAMERA_PARAMS_S
 * @brief     camera params.
 */
typedef struct rkAVS_CALIB_CAMERA_PARAMS_S
{
    uint32_t u32CameraNum;
    /**<camera numbers*/
    uint32_t u32CameraFov;
    /**<camea fov*/
    AVS_CALIB_CAMERA_TYPE_E eCameraType;                                /**
<camera type*/
} AVS_CALIB_CAMERA_PARAMS_S;
```

rkAVS_CALIB_MODEL_CAMERA_PARAMS_S 定义了相机的基本参数信息。

```
/*
 * @enum      AVS_STEREO_CALIB_CONFIG_S
 * @brief     input params.
 */
```

```

typedef struct AVS_STEREO_CALIB_CONFIG_S
{
    AVS_CALIB_IMAGE_DATA_S stImageData[MAX_CAMERA_NUM];
    /**<image params*/
    AVS_CALIB_BOARD_PARAMS_S stBoardParams;
    /**<board params*/
    AVS_CALIB_CAMERA_PARAMS_S stCameraParams; /* */
    <camera params*/
    uint32_t u32IsSaveCornersImage;
    /**<is save corners image*/
    uint32_t u32UseAccurateMode; /* */
    <optimize by stitch*/
    const char* pcTorrentPath;
    /**<model calibration result path*/
    const char* pcCalibResultPath;
    /**<calibration result path*/
}AVS_STEREO_CALIB_CONFIG_S;

```

AVS_STEREO_CALIB_CONFIG_S为双目产线标定的输入参数，包括图像信息、棋盘格信息、相机信息、是否保存角点的Flag、是否采用精确模式的flag、模型标定文件的路径以及最终标定结果的输出路径。

```

/*
 * @enum      rkAVS_STEREO_CALIB_POSE_S
 * @brief     output params.
 */
typedef struct rkAVS_STEREO_CALIB_POSE_S
{
    float f32RotationAngle[3];
    float f32Translation[3];
}AVS_STEREO_CALIB_POSE_S;

```

rkAVS_STEREO_CALIB_POSE_S定义了模组的位姿信息，包括旋转角度和位移量。

```

/*
 * @enum      rkAVS_STEREO_CALIB_RMS_S
 * @brief     output params.
 */
typedef struct rkAVS_STEREO_CALIB_RMS_S
{
    float f32LeftInternalRms;
    float f32RightInternalRms;
    float f32StereoExternalRms;
}AVS_STEREO_CALIB_RMS_S;

```

rkAVS_STEREO_CALIB_RMS_S定义了产线标定的误差，包括内参标定误差和外参标定误差，**该误差值越小越好，大于0.5效果较差。**

```

/*
 * @enum      rkAVS_STEREO_CALIB_CHECK_S
 * @brief    output params.
 */
typedef struct rkAVS_STEREO_CALIB_CHECK_S
{
    AVS_STEREO_CALIB_POSE_S stPose;
    AVS_STEREO_CALIB_RMS_S strms;
}AVS_STEREO_CALIB_CHECK_S;

```

`rkAVS_STEREO_CALIB_CHECK_S`作为双目产线标定的输出参数，用于判别标定效果的好坏。

```
int32_t rkAVS_getCalibVersion(char avsToolVersion[128]);
```

`rkAVS_getCalibVersion`函数用于获取算法库的版本信息。

```
int32_t rkAVS_calibProductStereo(AVS_STEREO_CALIB_CONFIG_S* psInputParams,  
AVS_STEREO_CALIB_CHECK_S* stStereoCheckParams);
```

rkAVS_calibProductStereo函数用于双目模组产线标定。

3.5.2 标定Demo

```
leftImagePath: ./calibProduct/src/left.yuv
rightImagePath: ./calibProduct/src/right.yuv
dstPath: ./calibProduct/rk_calib_result/
calibTorrentPath: ./calibProduct/src/rk_2_camera_result.xml
cameraNum: 2 //camera number
(2)
cameraType: 1 //0-pinhole、1-
omn、2-fisheye
cameraFov: 120 //horizontal fov
boardWidth: 15 //number of
chessboard corners in horizontal direction
boardHeight: 8 //number of
chessboard corners in vertical direction
squareSize: 30 //size of
chessboard(mm)
leftImageWidth: 1920 // camera0 image
width
leftImageHeight: 1080 // camera0 image
height
leftImageStride: 1920 // camera0 image
stride
rightImageWidth: 1920 // camera1 image
width
rightImageHeight: 1080 // camera1 image
height
rightImageStride: 1920 // camera1 image
stride
imageFormat: 2 //0-gray、1-rgb、2-
yuv
isSaveCornersImage: 1 //draw corners and
save
```

```
useAccurateMode: 1 //optimize by  
stitch
```

以上为产线标定时算法库的参考配置参数。

```
//保存产线标定筛选参数  
int saveCheckParams(std::string saveThreshPath, AVS_STEREO_CALIB_CHECK_S*  
stereoCalibCheck)  
{  
    std::ofstream fout(saveThreshPath);  
    if (!fout)  
    {  
        std::cout << "Open save_thresh_path fail!" << std::endl;  
        return 0;  
    }  
    else  
    {  
        fout << "leftInternalRms: " << stereoCalibCheck->stRms.f32LeftInternalRms << std::endl;  
        fout << "rightInternalRms: " << stereoCalibCheck->stRms.f32RightInternalRms << std::endl;  
        fout << "externalRms: " << stereoCalibCheck->stRms.f32StereoExternalRms  
<< std::endl;  
        fout << "euler_angles_x: " << stereoCalibCheck->stPose.f32RotationAngle[0] << std::endl;  
        fout << "euler_angles_y: " << stereoCalibCheck->stPose.f32RotationAngle[1] << std::endl;  
        fout << "euler_angles_z: " << stereoCalibCheck->stPose.f32RotationAngle[2] << std::endl;  
        fout << "t_x: " << stereoCalibCheck->stPose.f32Translation[0] <<  
std::endl;  
        fout << "t_y: " << stereoCalibCheck->stPose.f32Translation[1] <<  
std::endl;  
        fout << "t_z: " << stereoCalibCheck->stPose.f32Translation[2] <<  
std::endl;  
        //fout << "is_calib_succeed: " << is_succ << std::endl;  
    }  
    fout.close();  
    return 1;  
}
```

saveCheckParams用于保存产线标定的误差及模组信息，该参数可作为标定是否符合预期的判别参数。

```
bool readBinParams(std::string image_path, char* output_params, uint64_t  
length)  
{  
    std::ifstream fin;  
    //读取raw图  
    fin.open(image_path, std::ios::binary);  
    if (!fin) {  
        std::cerr << "open failed: " << image_path << std::endl;  
        return false;  
    }  
    // read函数读取（拷贝）流中的length各字节到buffer  
    fin.read((char*)output_params, length);  
    fin.close();
```

```
    return true;
}
```

readBinParams函数用于读取YUV数据。

```
//产线标定读取配置文件
int readCalibProductConfig(std::string readPath, std::vector<std::string>& stringData, std::vector<int32_t>& int32Data)
{
    std::string stringLine;
    std::ifstream ifs(readPath, std::ifstream::in);
    if (ifs.fail())
    {
        std::cout << "read calib product config fail!" << std::endl;
    }
    int32_t lineNumber = 0;
    int32_t valueData = 0;
    while (getline(ifs, stringLine))
    {
        std::stringstream ss(stringLine);
        std::string stringUnit;
        std::getline(ss, stringUnit, ' ');
        std::getline(ss, stringUnit, ' ');
        if (lineNumber < 4)
        {
            stringData.push_back(stringUnit);
        }
        else
        {
            valueData = atoi(stringUnit.c_str());
            int32Data.push_back(valueData);
        }
        lineNumber++;
    }
    ifs.close();
    return 0;
}
```

readCalibProductConfig函数用于读取产线标定的配置信息。

```
//产线标定Demo
int stereoCalibProdDemo(std::string configPath)
{
    std::vector<std::string> stringData;
    std::vector<int32_t> int32Data;
    readCalibProductConfig(configPath, stringData, int32Data);
    //获取版本信息
    char avsCalibVersion[128];
    rkAVS_getCalibVersion(avsCalibVersion);

    //初始化参数
    AVS_STEREO_CALIB_CONFIG_S stInputParams;
    //相机规格
    stInputParams.stCameraParams.u32CameraNum = int32Data[0];
    stInputParams.stCameraParams.u32CameraFov = int32Data[2];
    switch (int32Data[1])
```

```

{
    case 0:
        stInputParams.stCameraParams.eCameraType = AVS_CAMERA_TYPE_PINHOLE;
        break;
    case 1:
        stInputParams.stCameraParams.eCameraType = AVS_CAMERA_TYPE_OMN;
        break;
    case 2:
        stInputParams.stCameraParams.eCameraType = AVS_CAMERA_TYPE_FISHEYE;
        break;
    default:
        std::cout << "RK_AVSCALIB: CameraType error!" << std::endl;
        return 1;
}

//棋盘格规格
stInputParams.stBoardParams.u32BoardWidth = int32Data[3];
stInputParams.stBoardParams.u32BoardHeight = int32Data[4];
stInputParams.stBoardParams.u32SquareSize = int32Data[5];
//图像读取路径
std::string leftImagePath = stringData[0];
std::string rightImagePath = stringData[1];
//图像规格
uint8_t* leftImage;
uint8_t* rightImage;
cv::Mat leftImageMat, rightImageMat;
int32_t leftDataSize, rightDataSize;
for (int32_t cam = 0; cam < stInputParams.stCameraParams.u32CameraNum;
cam++)
{
    stInputParams.stImageData[cam].stImageParams.u32ImageWidth =
int32Data[6 + cam * 3];
    stInputParams.stImageData[cam].stImageParams.u32ImageHeight =
int32Data[7 + cam * 3];
    stInputParams.stImageData[cam].stImageParams.u32ImageStride =
int32Data[8 + cam * 3];
}
switch (int32Data[12])
{
case 0:
    stInputParams.stImageData[0].stImageParams.eImageFormat =
RK_AVSCALIB_IMAGE_FORMAT_TYPE_GRAY;
    stInputParams.stImageData[1].stImageParams.eImageFormat =
RK_AVSCALIB_IMAGE_FORMAT_TYPE_GRAY;
    leftImageMat = cv::imread(leftImagePath, 0);
    rightImageMat = cv::imread(rightImagePath, 0);
    leftImage = leftImageMat.ptr<uint8_t>(0);
    rightImage = rightImageMat.ptr<uint8_t>(0);
    break;
case 1:
    stInputParams.stImageData[0].stImageParams.eImageFormat =
RK_AVSCALIB_IMAGE_FORMAT_TYPE_RGB;
    stInputParams.stImageData[1].stImageParams.eImageFormat =
RK_AVSCALIB_IMAGE_FORMAT_TYPE_RGB;
    leftImageMat = cv::imread(leftImagePath, 1);
    rightImageMat = cv::imread(rightImagePath, 1);
    leftImage = leftImageMat.ptr<uint8_t>(0);
    rightImage = rightImageMat.ptr<uint8_t>(0);
    break;
}

```

```

    case 2:
        stInputParams.stImageData[0].stImageParams.eImageFormat =
RK_AVIS_IMAGE_FORMAT_TYPE_YUVNV12;
        stInputParams.stImageData[1].stImageParams.eImageFormat =
RK_AVIS_IMAGE_FORMAT_TYPE_YUVNV12;
        leftDataSize =
stInputParams.stImageData[0].stImageParams.u32ImageStride *
stInputParams.stImageData[0].stImageParams.u32ImageHeight * 3 / 2;
        rightDataSize =
stInputParams.stImageData[1].stImageParams.u32ImageStride *
stInputParams.stImageData[1].stImageParams.u32ImageHeight * 3 / 2;
        leftImageMat =
cv::Mat(stInputParams.stImageData[0].stImageParams.u32ImageHeight * 3 / 2,
stInputParams.stImageData[0].stImageParams.u32ImageStride, CV_8UC1);
        rightImageMat =
cv::Mat(stInputParams.stImageData[1].stImageParams.u32ImageHeight * 3 / 2,
stInputParams.stImageData[1].stImageParams.u32ImageStride, CV_8UC1);
        leftImage = leftImageMat.ptr<uint8_t>(0);
        rightImage = rightImageMat.ptr<uint8_t>(0);
        readBinParams(leftImagePath, (char*)leftImage, leftDataSize);
        readBinParams(rightImagePath, (char*)rightImage, rightDataSize);
        break;
    default:
        std::cout << "RK_AVIS_CALIB: ImageFormat error!" << std::endl;
        return 1;
}

//功能开关
stInputParams.u32IsSaveCornersImage = int32Data[13];
stInputParams.u32UseAccurateMode = int32Data[14];
//输入种子文件路径
stInputParams.pcTorrentPath = stringData[3].c_str();
//输入输出路径
std::string calibResultPath = stringData[2] + "rk_2_camera_result.xml";
stInputParams.pcCalibResultPath = calibResultPath.c_str();

//读取图像
stInputParams.stImageData[0].pu8ImageData = leftImage;
stInputParams.stImageData[1].pu8ImageData = rightImage;

//计时函数
#ifndef _WIN32
    clock_t startTime, endTime, timeAll;
    startTime = clock();
#else
    struct timeval startTime, endTime; // 秒+微妙
    time_t timeAll, timeAll_us;
    gettimeofday(&startTime, NULL);
#endif
/*********************************************产线拼接
*****************************************/
AVS_STEREO_CALIB_CHECK_S stStereoCheckParams;
int32_t status = rkAVS_calibProductStereo(&stInputParams,
&stStereoCheckParams);
if (status)
{
    printf("rkAVS_calibProductStereo fail!\n");
}

```

```

    else
    {
        printf("rkAVS_calibProductStrreo OK!\n");
    }
#endif _WIN32
endTime = clock();
timeAll = endTime - startTime;
printf("rkAVS_calibProductStrreo: %d ms\n", timeAll);
#else
gettimeofday(&endTime, NULL);
timeAll_us = 1000000 * (endTime.tv_sec - startTime.tv_sec) +
(endTime.tv_usec - startTime.tv_usec);
printf("rkAVS_calibProductStrreo: %d us\n", timeAll_us);
#endif

//保存产线筛选参数
std::string saveCheckPath = stringData[2] + "productCheckParams.txt";
saveCheckParams(saveCheckPath, &stStereoCheckParams);
return 0;
}

```

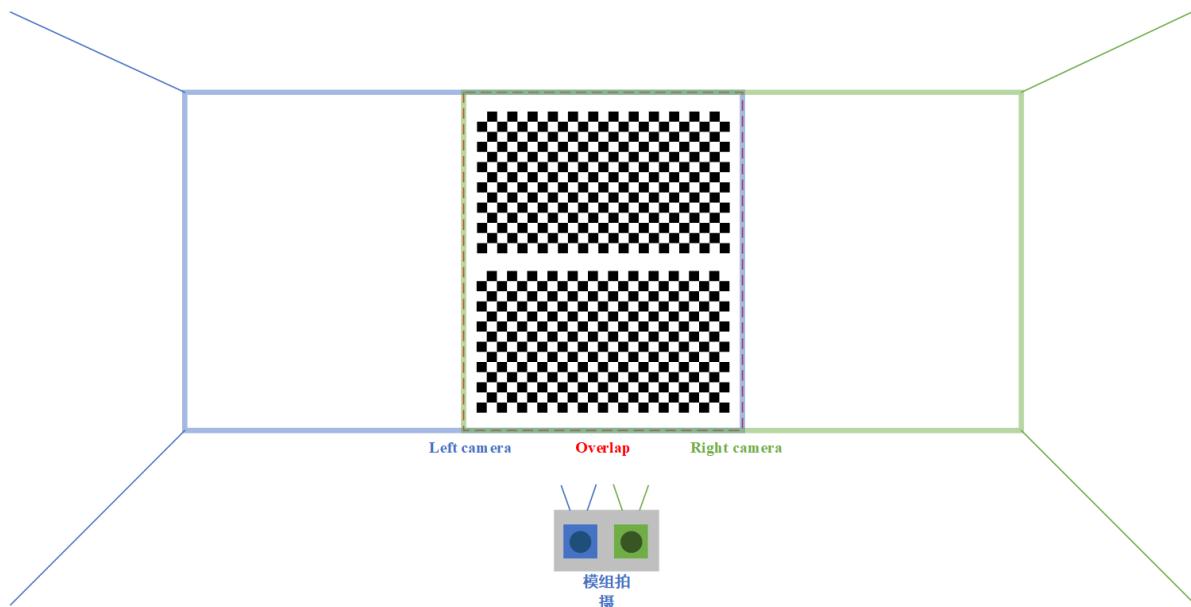
calibProdDemo函数为产线标定调用算法库完成标定的Demo。

4. 模组验证

模组验证其目标就是提前筛选出问题模组，提高出厂良率。验证过程主要利用模组标定参数（标定输出的xml），对于指定验证chart板进行验证，利用检测结果与实际值进行对比，进而筛选出有问题的模组。

4.1 操作环境

图4.1 验证环境示意图



如图4.1所示，为模组验证时的场景示意图。每次拍摄标定板时，要求模组的两个相机能够同时拍摄到两块完整的棋盘格。

4.2 棋盘格规格设计

每个棋盘格的规格与标定场景一致，具体可参考3.3。

4.3 API接口及Demo

4.3.1 接口定义

```
int32_t rkAVS_verifyProductStereo(AVS_STEREO_CALIB_CONFIG_S* psInputParams,  
float* stereoRms);
```

rkAVS_verifyProductStereo函数用于验证标定参数的准确性，输出结果stereoRms越小越好，一般要求小于0.5。

4.3.2 验证Demo

```
//产线标定验证Demo  
int stereoVerifyProdDemo(std::string configPath)  
{  
    std::vector<std::string> stringData;  
    std::vector<int32_t> int32Data;  
    readCalibProdictConfig(configPath, stringData, int32Data);  
    //获取版本信息  
    char avsCalibVersion[128];  
    rkAVS_getCalibVersion(avscalibVersion);  
  
    //初始化参数  
    AVS_STEREO_CALIB_CONFIG_S stInputParams;  
    //相机规格  
    stInputParams.stCameraParams.u32CameraNum = int32Data[0];  
    stInputParams.stCameraParams.u32CameraFov = int32Data[2];  
    switch (int32Data[1])  
    {  
        case 0:  
            stInputParams.stCameraParams.eCameraType = AVS_CAMERA_TYPE_PINHOLE;  
            break;  
        case 1:  
            stInputParams.stCameraParams.eCameraType = AVS_CAMERA_TYPE_OMN;  
            break;  
        case 2:  
            stInputParams.stCameraParams.eCameraType = AVS_CAMERA_TYPE_FISHEYE;  
            break;  
        default:  
            std::cout << "RK_AVSCALIB: CameraType error!" << std::endl;  
            return 1;  
    }  
    //棋盘格规格  
    stInputParams.stBoardParams.u32BoardWidth = int32Data[3];  
    stInputParams.stBoardParams.u32BoardHeight = int32Data[4];  
    stInputParams.stBoardParams.u32SquareSize = int32Data[5];  
    //图像读取路径  
    std::string leftImagePath = stringData[0];  
    std::string rightImagePath = stringData[1];  
    //图像规格  
    uint8_t* leftImage;  
    uint8_t* rightImage;
```

```

cv::Mat leftImageMat, rightImageMat;
int32_t leftDataSize, rightDataSize;
for (int32_t cam = 0; cam < stInputParams.stCameraParams.u32CameraNum;
cam++)
{
    stInputParams.stImageData[cam].stImageParams.u32ImageWidth =
int32Data[6 + cam * 3];
    stInputParams.stImageData[cam].stImageParams.u32ImageHeight =
int32Data[7 + cam * 3];
    stInputParams.stImageData[cam].stImageParams.u32ImageStride =
int32Data[8 + cam * 3];
}
switch (int32Data[12])
{
case 0:
    stInputParams.stImageData[0].stImageParams.eImageFormat =
RK_AVIS_IMAGE_FORMAT_TYPE_GRAY;
    stInputParams.stImageData[1].stImageParams.eImageFormat =
RK_AVIS_IMAGE_FORMAT_TYPE_GRAY;
    leftImageMat = cv::imread(leftImagePath, 0);
    rightImageMat = cv::imread(rightImagePath, 0);
    leftImage = leftImageMat.ptr<uint8_t>(0);
    rightImage = rightImageMat.ptr<uint8_t>(0);
    break;
case 1:
    stInputParams.stImageData[0].stImageParams.eImageFormat =
RK_AVIS_IMAGE_FORMAT_TYPE_RGB;
    stInputParams.stImageData[1].stImageParams.eImageFormat =
RK_AVIS_IMAGE_FORMAT_TYPE_RGB;
    leftImageMat = cv::imread(leftImagePath, 1);
    rightImageMat = cv::imread(rightImagePath, 1);
    leftImage = leftImageMat.ptr<uint8_t>(0);
    rightImage = rightImageMat.ptr<uint8_t>(0);
    break;
case 2:
    stInputParams.stImageData[0].stImageParams.eImageFormat =
RK_AVIS_IMAGE_FORMAT_TYPE_YUVNV12;
    stInputParams.stImageData[1].stImageParams.eImageFormat =
RK_AVIS_IMAGE_FORMAT_TYPE_YUVNV12;
    leftDataSize =
stInputParams.stImageData[0].stImageParams.u32ImageStride *
stInputParams.stImageData[0].stImageParams.u32ImageHeight * 3 / 2;
    rightDataSize =
stInputParams.stImageData[1].stImageParams.u32ImageStride *
stInputParams.stImageData[1].stImageParams.u32ImageHeight * 3 / 2;
    leftImageMat =
cv::Mat(stInputParams.stImageData[0].stImageParams.u32ImageHeight * 3 / 2,
stInputParams.stImageData[0].stImageParams.u32ImageStride, CV_8UC1);
    rightImageMat =
cv::Mat(stInputParams.stImageData[1].stImageParams.u32ImageHeight * 3 / 2,
stInputParams.stImageData[1].stImageParams.u32ImageStride, CV_8UC1);
    leftImage = leftImageMat.ptr<uint8_t>(0);
    rightImage = rightImageMat.ptr<uint8_t>(0);
    readBinParams(leftImagePath, (char*)leftImage, leftDataSize);
    readBinParams(rightImagePath, (char*)rightImage, rightDataSize);

break;
default:

```

```

        std::cout << "RK_AVSCALIB: ImageFormat error!" << std::endl;
        return 1;
    }

//功能开关
stInputParams.u32IsSaveCornersImage = int32Data[13];
stInputParams.u32UseAccurateMode = int32Data[14];
//输入种子文件路径
stInputParams.pcTorrentPath = NULL;
//输入输出路径
std::string calibResultPath = stringData[2] + "rk_2_camera_result.xml";
//标定参数所在的输入路径
stInputParams.pcCalibResultPath = calibResultPath.c_str();

//读取图像
stInputParams.stImageData[0].pu8ImageData = leftImage;
stInputParams.stImageData[1].pu8ImageData = rightImage;
//模型标定

//计时函数
#ifndef _WIN32
    clock_t startTime, endTime, timeAll;
    startTime = clock();
#else
    struct timeval startTime, endTime; // 秒+微妙
    time_t timeAll, timeAll_us;
    gettimeofday(&startTime, NULL);
#endif
/*******************************************产线拼接
*****************************************/
float stVerifyRms;
int32_t status = rkAVS_verifyProductStereo(&stInputParams, &stVerifyRms);
if (status)
{
    printf("rkAVS_calibProductStereo fail!\n");
}
#endif _WIN32
endTime = clock();
timeAll = endTime - startTime;
printf("rkAVS_verifyProductStereo: %d ms\n", timeAll);
#else
gettimeofday(&endTime, NULL);
timeAll_us = 1000000 * (endTime.tv_sec - startTime.tv_sec) +
(endTime.tv_usec - startTime.tv_usec);
printf("rkAVS_verifyProductStereo: %d us\n", timeAll_us);
#endif
printf("verify rms is: %f", stVerifyRms);
return 0;
}

```

5. 标定异常分析

5.1 RK_AVSCALIB_STATUS_INVALID_PARAM

算法库返回异常值3，表示输入参数有问题，请检查图像输入尺寸、棋盘格参数及其他配置参数是否符合要求。

5.2 RK_AVSCALIB_STATUS_EOF

算法库返回异常值-1，表示算法库内部计算错误，主要分为以下几种情况。

5.2.1 Split chessboard error

输入单帧图像内包含多个棋盘格，内部标定时需要区分多个棋盘格所在位置，该错误表明算法检查多个棋盘格时出现错误。遇到该问题可通过以下几种方式解决：

- 尽可能保证拍摄背景为纯白或者纯黑（纹理较少），如在背景为白墙的位置拍摄标定板或者在标定板后面布置一块白色或者黑色的幕布、板子；
- 通过调节ISP保证输入图像整体亮度均匀，不可存在局部过曝、过暗等情况。

5.2.2 Good chessboard number < n

输入的图像内包含多个棋盘格，如果内部算法检测到的棋盘格个数少于指定棋盘格个数，算法会报该错误并返回-1。遇到该问题可通过以下几种方式解决：

- 保证所需棋盘格（标定每个相机为4个棋盘格，验证为2个棋盘格）都在相机视野内，不可存在拍摄不完整的棋盘格；
- 调节ISP保证输入图像整体亮度均匀，不可存在局部过曝、过暗的情况,如图5.1所示；
- 保证多个棋盘格之间有足够的距离，彼此不可靠的太近；
- 拍摄棋盘格应该清晰，不能出现某个棋盘格模糊的情况，如图5.2右侧两个棋盘格所示；
- 四个棋盘格应分布在对应的辅助框内，如图5.3所示。

图5.1 局部过曝模糊示例

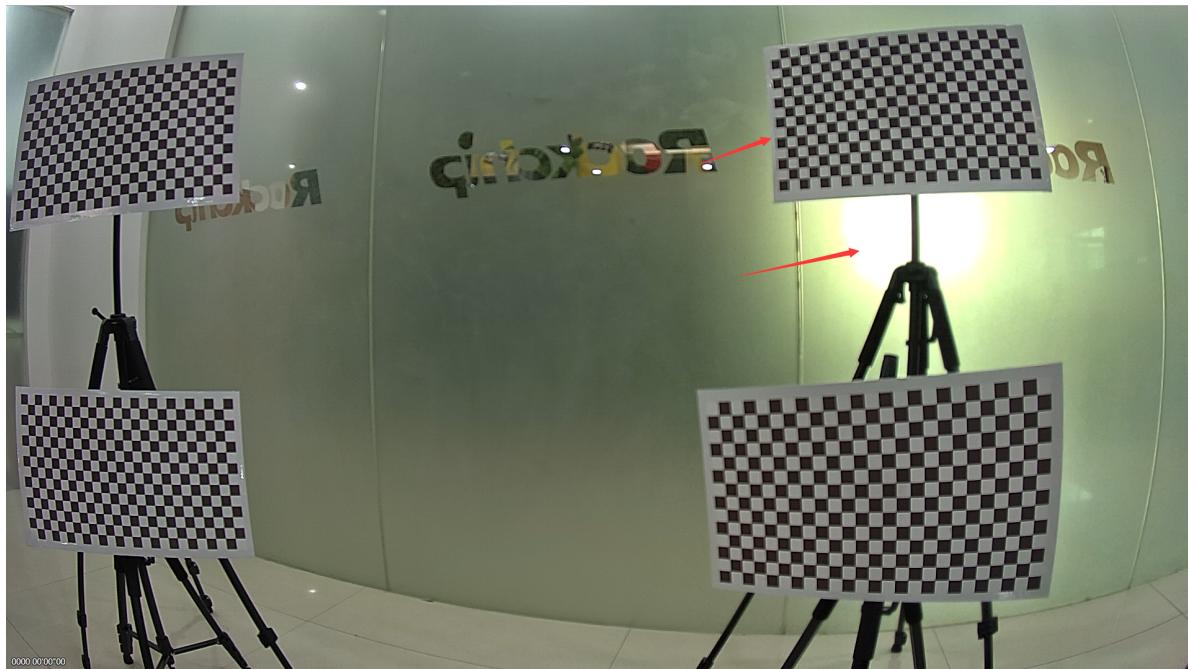
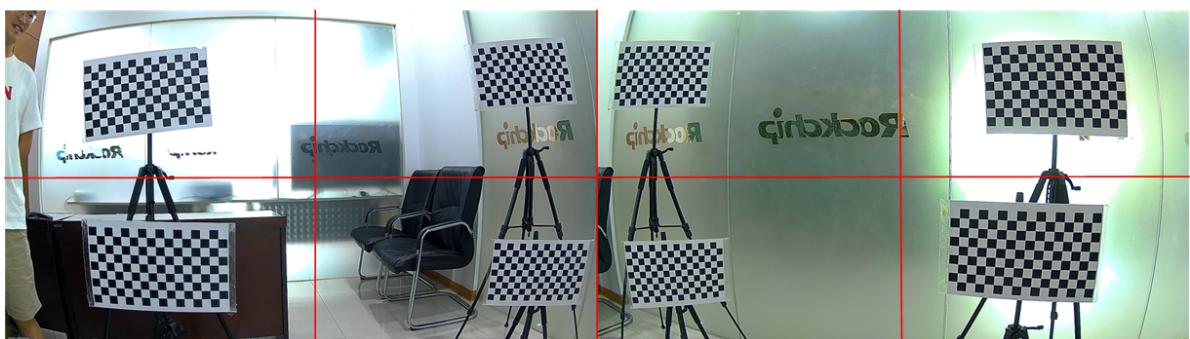


图5.2 棋盘格模糊示例



图5.3 棋盘格摆放位置说明



可以通过检查保存下来的角点图判断哪个棋盘格没有检测到，如图5.4所示。

图5.4 棋盘格角点提取图



如图5.4所示，右上角的棋盘格因为周边过曝，导致角点提取失败。

5.3 标定精度低

5.3.1 问题分析

该问题主要体现在标定输出的RMS较大，可通过调节标定板的摆放优化标定结果。

- 按照3.2中的场景布局布置棋盘格，保证每个棋盘格角点能够拍摄清晰；
- 查看图像是否存在反光、过曝、过暗、噪点多的情况，改善拍图质量；
- 重叠区域的两个棋盘格所在位置上下不够对称也会影响精度；
- 可采用背光板提高标定精度。

图5.1 标定板拍摄示例图

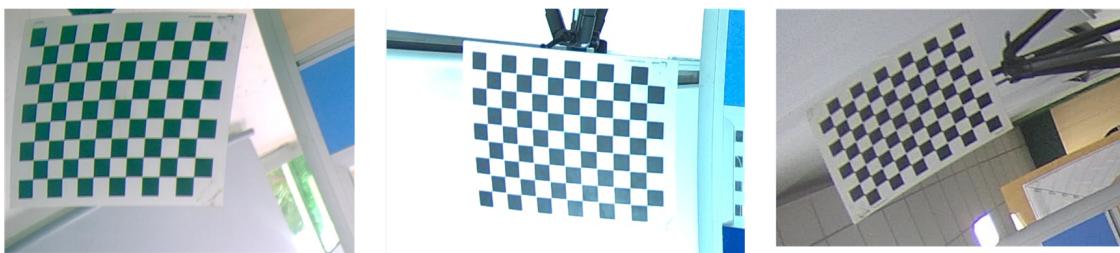


5.3.2 错误示例

- 反光



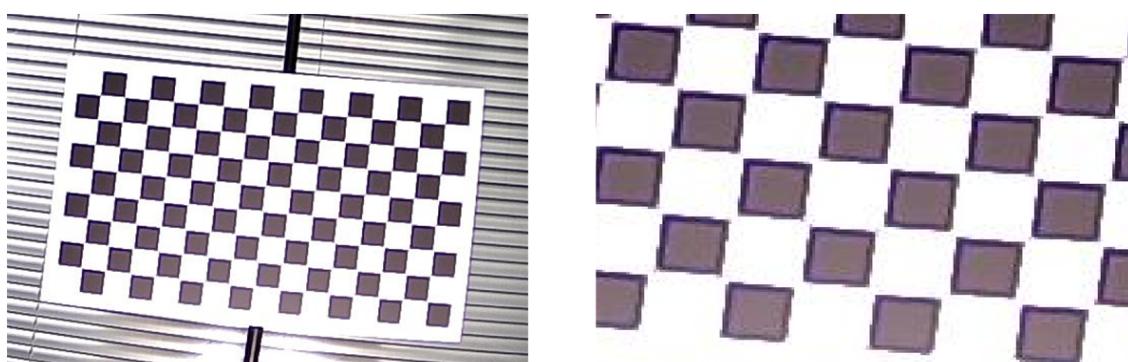
- 棋盘格模糊



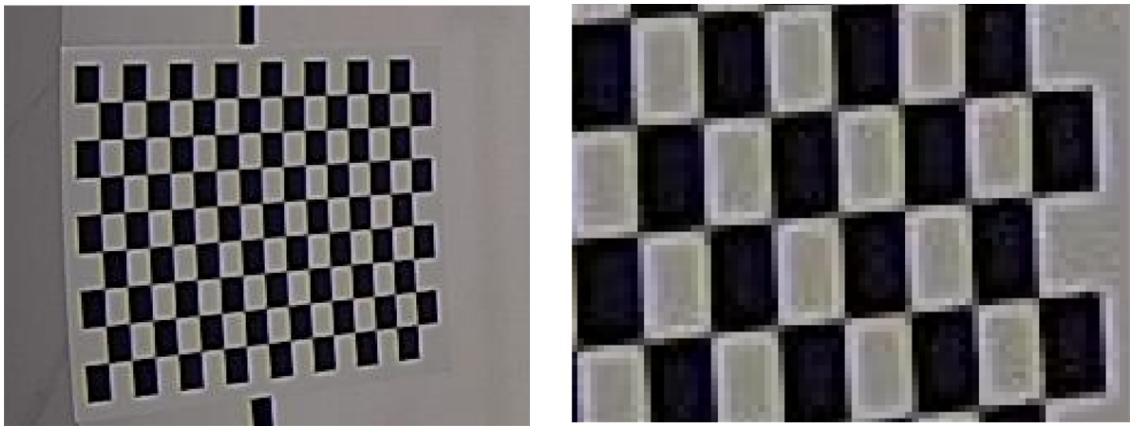
- 棋盘格不平整

张贴棋盘格不平整，存在鼓包等现象。

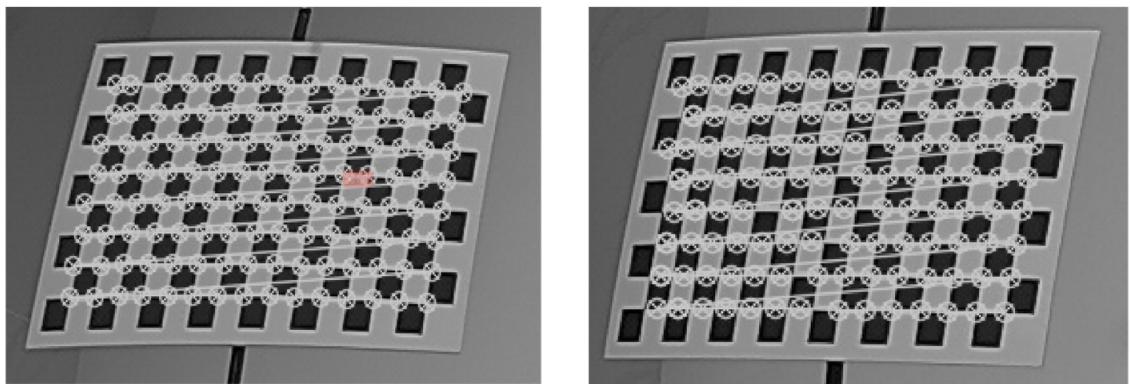
- 棋盘格过曝



- 光照不足



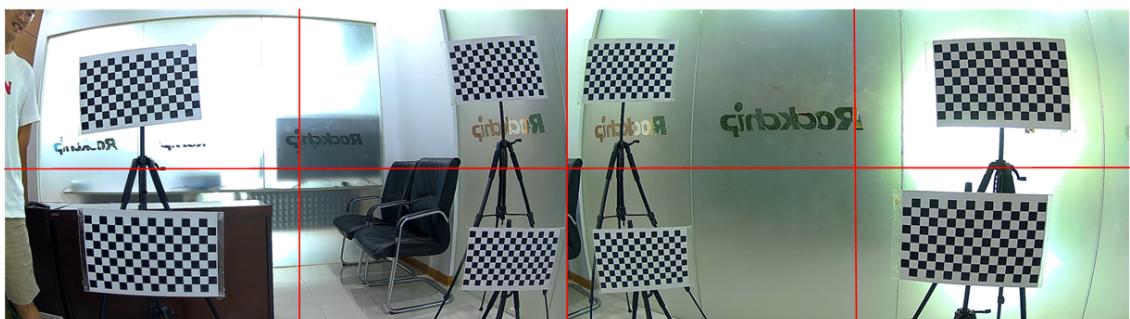
- 角点提取出错



- 重叠区域棋盘格上下不对称



如上图所示，中间重叠区域的两个棋盘格相对于中线不够上下对称，也会影响标定精度，最好可以像下图所示进行摆放。



- 拍摄时相机或者棋盘格存在震动