

Rockchip 双目拼接模型标定指南

发布版本：V1.1.2

日期：2023.07.10

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 © 2023 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模型标定基本原理、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
2023.06.20	V1.1.1	mingpei.liu	初始版本
2023.07.10	V1.1.2	mingpei.liu	完善拍图注意事项

Rockchip 双目拼接模型标定指南

1. 概述
 - 1.1 双目拼接概述
 - 1.2 模型标定概述
2. 模型标定
 - 2.1 棋盘格规格设计
 - 2.2 内参拍图
 - 2.3 外参拍图
 - 2.4 内外参标定
3. API接口及Demo
 - 3.1 接口定义
 - 3.2 标定Demo
 - 3.3 角点提取Demo
4. 标定失败及精度差的原因
 - 4.1 拍图不符合要求
 - 4.1.1 标定板反光
 - 4.1.2 标定板未拍全
 - 4.1.3 图像模糊
 - 4.1.4 棋盘格不平整
 - 4.1.5 棋盘格过曝
 - 4.1.6 光照不足
 - 4.1.7 角点提取出错
 - 4.2 参数配置错误

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幅图像，从而实现获取更大视野的需求。

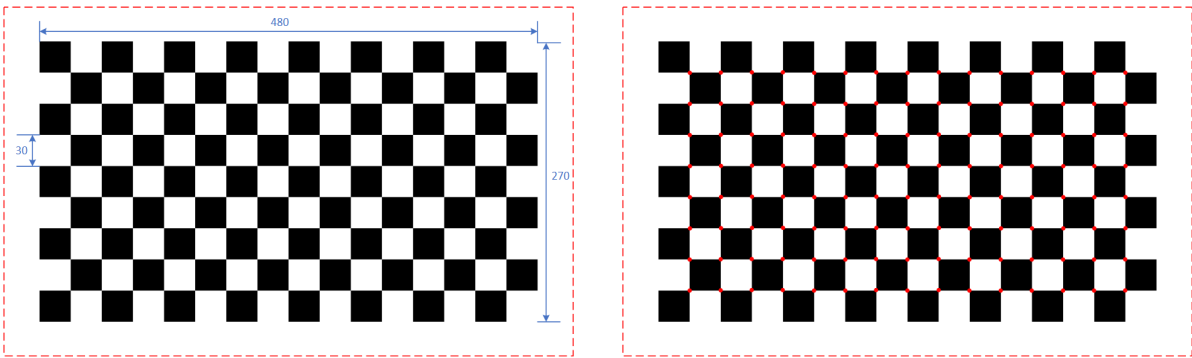
1.2 模型标定概述

对于结构固定的产品，模型标定只需要进行一次，可以在同一批次产品选择一个结构最佳的机器进行模型标定。标定时需要拍摄大量棋盘格图像，然后经过AVS模型标定工具生成标定文件。标定参数包括左右镜头的各自的内参、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. 模型标定

2.1 棋盘格规格设计

图2-1棋盘格规格设计



如图2-1为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：

- 1) 棋盘格的长为480mm，宽为270mm；
- 2) 棋盘格中每个方块的物理尺寸是30mm，精度要求为0.1mm；
- 3) 棋盘格要求表面平整；
- 4) 棋盘格的长边共16个方格，短边共9个方格；
- 5) 图中红色圈所标出的为内点，共15*8个内点。

注意：棋盘格四周须有留白区域，如图2-1所示，棋盘格四周与红色边框之间有留白区域，留白区域的边长大于一个棋盘格即可。

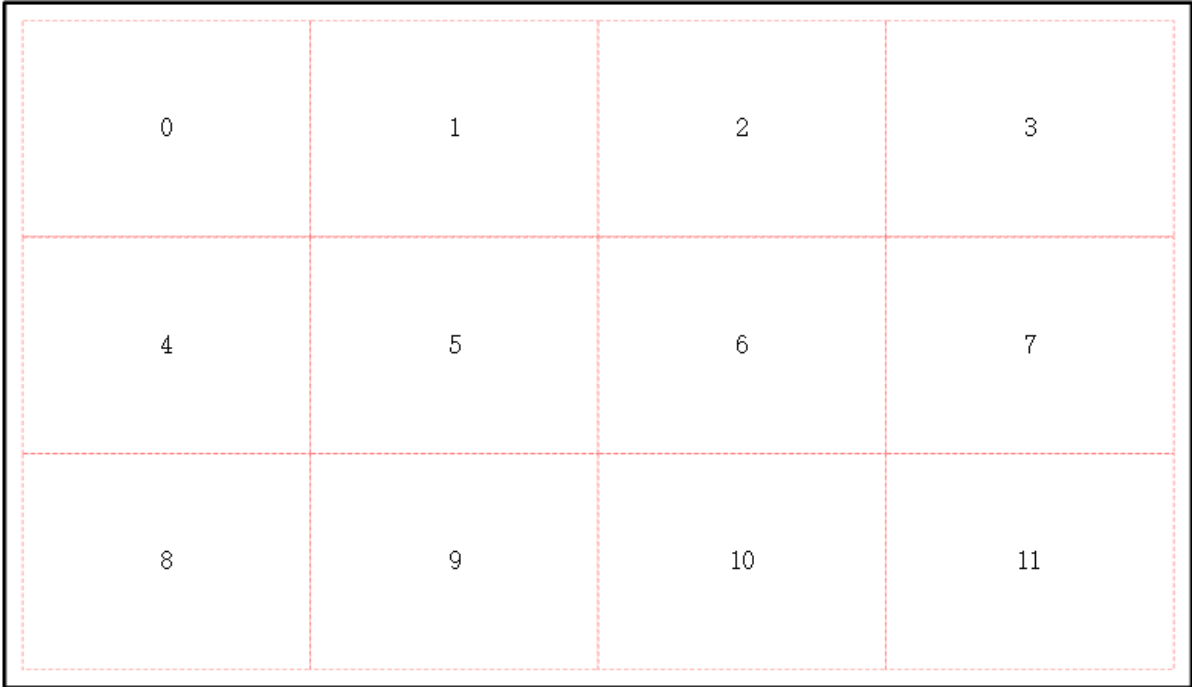
以上为设计案例，实际使用中尺寸及棋盘格个数可自由设计。

2.2 内参拍图

该场景用于标定单个相机的内参（焦距、光心和畸变系数），具体拍图要求如下：

- 1) 每个镜头至少拍取 10 张图像。保存时以<固定前缀名><镜头号>_<帧序号><格式后缀> 来对图像进行命名；
- 2) 拍图时，将棋盘格置于镜头前方，不断变换棋盘格方向、距离，改变其在视野中的位置，尽可能的使棋盘格覆盖视野中的所有位置；
- 3) 对于非鱼眼镜头，近距离棋盘格约占画面 1/4 即可，远距离约占画面 1/8；对于鱼眼镜头，由于 FOV 较大，近距离棋盘格约占画面 1/5，远距离约占画面1/10 左右，抓拍时建议将棋盘格均匀绕鱼眼镜头一周，采集不同位置的棋盘格图像。

图2-2 内参标定棋盘格拍摄分布示意图



如图2-2所示，为内参拍摄的示意图，拍摄时尽可能的让棋盘格覆盖相机视野。实际拍摄效果图如图2-3所示。

图2-3 内参标定实拍图



拍图命名规则为: "camera"+相机编号+"_" +图像序号（3位数，不足3位数的前面补0），例如0号相机第1幅图为“camera0_000”，0号相机第2幅图为“camera0_001”，2号相机第0幅图为“camera2_000”，2号相机第10幅图为“camera2_009”，保存格式为".jpg",".yuv",".bin"等常用格式。

拍图注意事项

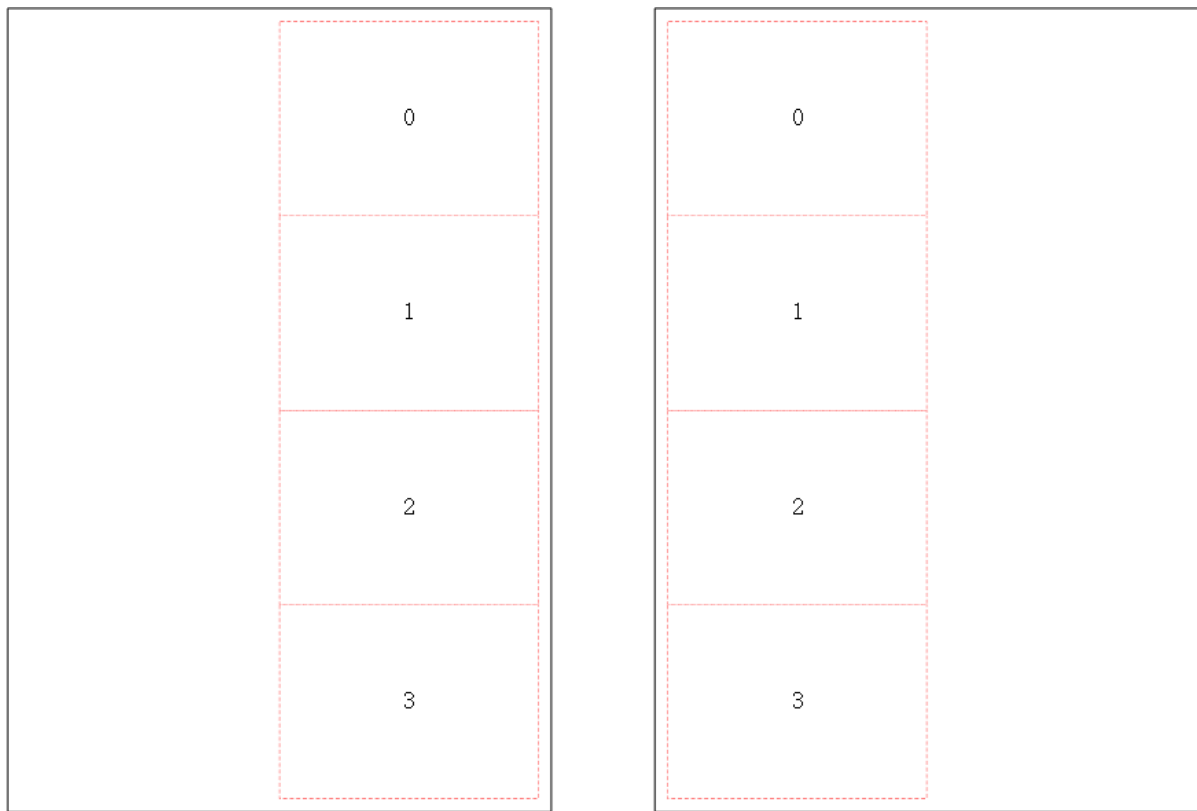
- 1) 拍摄棋盘格图片必须清晰，不能模糊；
- 2) 整个拍摄过程中，相机需保持定焦，不能随意调焦，一旦调节，整个标定过程需重新进行；
- 3) 标定环境需光照充足且均匀；
- 4) 拍摄过程中，视野内只能有一个棋盘格；
- 5) 拍摄过程中棋盘格不能存在反光现象；
- 6) 拍摄到的棋盘格需完整，不能存在缺失现象；
- 7) 拍摄的棋盘格不能存在过曝或者过暗的情况；
- 8) 棋盘格需覆盖不同角度、不同区域和不同距离。

2.3 外参拍图

该场景用于标定相邻两个相机间的外参（旋转量和平移量），具体拍图要求如下：

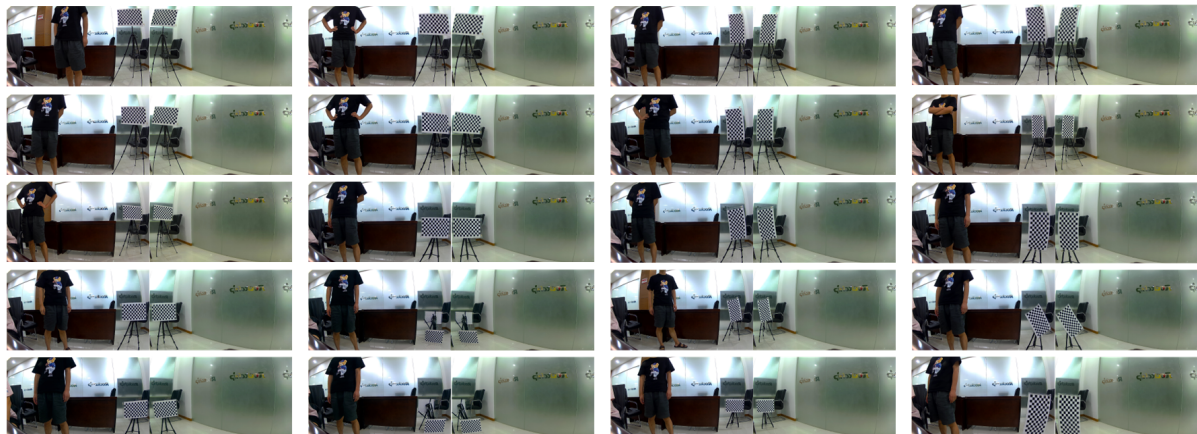
- 1) 每组镜头（相邻两个镜头）至少抓拍 10 对图片。保存时以<固定前缀名><镜头号>_<较小镜头号><较大镜头号>_<帧序号><格式后缀> 来对图像进行命名。
- 2) 拍图时，首选确定是哪两个相机拍摄，保证两个相机同时拍摄同一个棋盘格。棋盘格的摆放位置应全部位于两个相机的重叠区域范围内，并在该范围移动，尽可能地覆盖全部重叠区域，拍摄过程中应对棋盘格做旋转和远近距离的改变，保证最少涵盖三种拍摄距离。
- 3) 在移动棋盘格的过程中应保证棋盘格能够清晰拍到。
- 4) 拍摄过程中需要保证两个相机同步出图，模组静止，不可有震动、晃动等干扰模组状态的情况出现。

图2-4外参标定棋盘格拍摄分布示意图



如图2-4所示，拍摄外参标定所需的棋盘格图像时，需保证左右两个相机同时拍摄到完整的棋盘格。实际的拍摄效果如图3-5所示。

图2-5 外参标定实拍图



拍图命名规则为：“camera”+ 相机编号 + “_”+ 两两相机的编号 + “_”+ 图像序号（3位数，不足3位数的前面补0），例如0号相机与1号相机拍摄到的第5幅图为：“camera0_01_004”和“camera1_01_004”，保存格式为“.jpg", ".yuv", ".bin"等常用格式。

拍图注意事项

- 1) 拍摄棋盘格图片必须清晰，不能模糊；
- 2) 整个拍摄过程中，相机需保持定焦，不能随意调焦，一旦调节，整个标定过程需重新进行；
- 3) 标定环境需光照充足且均匀；
- 4) 拍摄过程中，视野内只能有一个棋盘格；
- 5) 拍摄过程中棋盘格不能存在反光现象；
- 6) 拍摄到的棋盘格需完整，不能存在缺失现象；
- 7) 拍摄的棋盘格不能存在过曝或者过暗的情况；

- 8) 拍摄过程中需要保证两个相机同步出图，模组保持静止；
- 9) 棋盘格需覆盖不同角度、不同区域和不同距离。

2.4 内外参标定

将拍摄好的棋盘格图像放入指定文件夹，通过AVS标定工具进行标定，获取标定后的文件。

3. API接口及Demo

3.1 接口定义

```
/*
 * @enum      rkAVS_CALIB_STATUS_E
 * @brief     the return status of function
 */
typedef enum rkAVS_CALIB_STATUS_E
{
    RK_AVS_CALIB_STATUS_EOF = -1,           /**< error of function inside */
    RK_AVS_CALIB_STATUS_OK = 0,             /**< run successfully */
    RK_AVS_CALIB_STATUS_FILE_READ_ERROR,    /**< error: fail to read file */
    RK_AVS_CALIB_STATUS_FILE_WRITE_ERROR,   /**< error: fail to write file */
    RK_AVS_CALIB_STATUS_INVALID_PARAM,      /**< error: invalid parameter */
    RK_AVS_CALIB_STATUS_ALLOC_FAILED,       /**< error: fail to alloc buffer */
    RK_AVS_CALIB_STATUS_BUTT                /**< reserved fields */
}AVS_CALIB_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_AVS_IMAGE_FORMAT_TYPE_GRAY = 0,      /**<gray format*/
    RK_AVS_IMAGE_FORMAT_TYPE_RGB = 1,       /**<rgb format*/
    RK_AVS_IMAGE_FORMAT_TYPE_YUVNV12 = 2,   /**<yuvnv12
format*/
} AVS_CALIB_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*/
    AVS_CALIB_IMAGE_FORMAT_E eImageFormat;  /**<image format*/
}AVS_CALIB_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_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;                                /**
<camera 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    rkAVS_CALIB_MODEL_CONFIG_S
 * @brief   input params.
 */
typedef struct rkAVS_CALIB_MODEL_CONFIG_S
{
    AVS_CALIB_IMAGE_PARAMS_S stImageParams;
    /**<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* pcInputPath;                                /**
<calibration images path*/
    const char* pcOutputPath;                              /**
<calibration result path*/
    const char* pcImageNameFormat;                          /**
<image name format .yuv, .png, .jpg, .bmp*/
}AVS_CALIB_MODEL_CONFIG_S;

```

AVS_CALIB_MODEL_CONFIG_S为模型标定的输入参数，包括图像信息、棋盘格信息、相机信息、是否保存角点的Flag、输入图像所在文件夹路径、最终标定结果的输出路径以及输入图像的后缀名。

```

/*
 * @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作为标定的输出参数，用于判别标定效果的好坏，模型标定返回的为两个相机间的最大重投影误差和相应姿态。

```

/*
 * @enum    rkAVS_CALIB_FIND_POINTS_S
 * @brief   input params.
 */
typedef struct rkAVS_CALIB_FIND_POINTS_S
{
    AVS_CALIB_IMAGE_DATA_S stImageData;
    /**<image params*/
    AVS_CALIB_BOARD_PARAMS_S stBoardParams;
    /**<board params*/
    const char* pcPointsImagePath;
    /**<points image save path*/
}AVS_CALIB_FIND_POINTS_S;

```

rkAVS_CALIB_FIND_POINTS_S作为角点提取检测的输入参数，用于模型标定时验证拍摄的图像是否可以准确的检测到棋盘格角点。

```
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_findPoints(AVS_CALIB_FIND_POINTS_S* psInputParams);
```

rkAVS_findPoints函数用于获取输入图像的角点图像。

```
int32_t rkAVS_calibModel(AVS_CALIB_MODEL_CONFIG_S* psInputParams,
    AVS_STEREO_CALIB_CHECK_S* stStereoCheckParams);
```

rkAVS_calibModel函数用于模型标定。

3.2 标定Demo

```
srcPath: ./calibModel/all_calib_images/
dstPath: ./calibModel/rk_calib_result/
imageNameFormat: .yuv
    // .jpg、.png、.bmp、.bin、.yuv
cameraNum: 2                                // camera number (2-8)
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)
imageWidth: 1920                                // image width
imageHeight: 1080                                // image height
imageStride: 1920                                // image stride
imageFormat: 2                                // 0-gray, 1-rgb, 2-yuv
isSaveCornersImage: 0                            // draw corners and save
useAcurateMode: 0                                // Optimize by Stitch
```

以上为模型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。

```

//模型标定读取配置文件
int readCalibModelConfig(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 model config fail!" << std::endl;
    }
    int32_t lineNum = 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 (lineNum < 3)
        {
            stringData.push_back(stringUnit);
        }
        else
        {
            valueData = atoi(stringUnit.c_str());
            int32Data.push_back(valueData);
        }
        lineNum++;
    }
    ifs.close();
    return 0;
}

```

readCalibModelConfig函数用于读取模型标定的配置信息。

```

//模型标定Demo
int calibModelDemo(std::string configPath)
{
    std::vector<std::string> stringData;
    std::vector<int32_t> int32Data;
    readCalibModelConfig(configPath, stringData, int32Data);
    std::string input_calib_image_path = stringData[0]; //标定图像所在路径
    std::string output_calib_result_path = stringData[1] +
    "rk_2_camera_result.xml";
    std::string input_image_format = stringData[2]; //标定图像后缀名
    //获取版本信息
    char avsCalibVersion[128];
    rkAVS_getCalibVersion(avsCalibVersion);
    //初始化参数
    AVS_CALIB_MODEL_CONFIG_S stInputParams;
    //相机规格
    stInputParams.stCameraParams.u32CameraFov = int32Data[2];
    stInputParams.stCameraParams.u32CameraNum = int32Data[0];
    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_AVS_CALIB: CameraType error!" << std::endl;
        return 1;
    }
    //棋盘格规格
    stInputParams.stBoardParams.u32BoardHeight = int32Data[4];
    stInputParams.stBoardParams.u32BoardWidth = int32Data[3];
    stInputParams.stBoardParams.u32SquareSize = int32Data[5];
}

```

```

//图像规格
stInputParams.pcImageNameFormat = input_image_format.c_str();
switch (int32Data[9])
{
case 0:
    stInputParams.stImageParams.eImageFormat = RK_AVS_IMAGE_FORMAT_TYPE_GRAY;
    break;
case 1:
    stInputParams.stImageParams.eImageFormat = RK_AVS_IMAGE_FORMAT_TYPE_RGB;
    break;
case 2:
    stInputParams.stImageParams.eImageFormat =
RK_AVS_IMAGE_FORMAT_TYPE_YUVNV12;
    break;
default:
    std::cout << "RK_AVS_CALIB: ImageFormat error!" << std::endl;
    return 1;
}
stInputParams.stImageParams.u32ImageHeight = int32Data[7];
stInputParams.stImageParams.u32ImageWidth = int32Data[6];
if (stInputParams.stImageParams.eImageFormat == RK_AVS_IMAGE_FORMAT_TYPE_RGB)
    stInputParams.stImageParams.u32ImageStride =
stInputParams.stImageParams.u32ImageWidth * 3;
else
    stInputParams.stImageParams.u32ImageStride = int32Data[8];
//输入输出路径
stInputParams.pcInputPath = input_calib_image_path.c_str();
stInputParams.pcOutputPath = output_calib_result_path.c_str();
//功能开关
stInputParams.u32IsSaveCornersImage = int32Data[10];
stInputParams.u32UseAccurateMode = int32Data[11];
//模型标定
AVS_STEREO_CALIB_CHECK_S stStereoCheckParams;
rkAVS_calibModel(&stInputParams, &stStereoCheckParams);
//保存产线筛选参数
std::string saveCheckPath = stringData[1] + "productCheckParams.txt";
saveCheckParams(saveCheckPath, &stStereoCheckParams);
return 0;
}

```

calibModelDem函数为模型标定调用算法库完成标定的Demo。

3.3 角点提取Demo

```
srcPath: ./camera0_01_000.yuv
dstPath: ./camera0_01_000.jpg
boardWidth: 15 //number of
chessboard corners in horizontal direction
boardHeight: 8 //number of
chessboard corners in vertical direction
squareSize: 30 //size of
chessboard(mm)
imageWidth: 1920 //image width
imageHeight: 1080 //image height
imageStride: 1920 //image stride
imageFormat: 2 //0-gray、1-rgb、2-
yuv
```

以上为角点提取的配置参数。

```
//角点提取读取配置文件
int readFindPointsConfig(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 lineNum = 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 (lineNum < 2)
        {
            stringData.push_back(stringUnit);
        }
        else
        {
            valueData = atoi(stringUnit.c_str());
            int32Data.push_back(valueData);
        }
        lineNum++;
    }
    ifs.close();
    return 0;
}
```

readFindPointsConfig函数用于读取角点提取的配置文件。

```

//角点提取Demo
int findPointsDemo(std::string configPath)
{
    std::vector<std::string> stringData;
    std::vector<int32_t> int32Data;
    readFindPointsConfig(configPath, stringData, int32Data);
    //获取版本信息
    char avsCalibVersion[128];
    rkAVS_getCalibVersion(avsCalibVersion);

    //初始化参数
    AVS_CALIB_FIND_POINTS_S stInputParams;
    //棋盘格规格
    stInputParams.stBoardParams.u32BoardWidth = int32Data[0];
    stInputParams.stBoardParams.u32BoardHeight = int32Data[1];
    stInputParams.stBoardParams.u32SquareSize = int32Data[2];
    //图像读取路径
    std::string srcImagePath = stringData[0];
    std::string dstImagePath = stringData[1];
    //图像规格
    uint8_t* pu8SrcImage;
    cv::Mat imageMat;
    int32_t dataSize;
    stInputParams.stImageData.stImageParams.u32ImageWidth = int32Data[3];
    stInputParams.stImageData.stImageParams.u32ImageHeight = int32Data[4];
    stInputParams.stImageData.stImageParams.u32ImageStride = int32Data[5];

    switch (int32Data[6])
    {
    case 0:
        stInputParams.stImageData.stImageParams.eImageFormat =
RK_AVS_IMAGE_FORMAT_TYPE_GRAY;
        imageMat = cv::imread(srcImagePath, 0);
        pu8SrcImage = imageMat.ptr<uint8_t>(0);
        break;
    case 1:
        stInputParams.stImageData.stImageParams.eImageFormat =
RK_AVS_IMAGE_FORMAT_TYPE_RGB;
        imageMat = cv::imread(srcImagePath, 1);
        pu8SrcImage = imageMat.ptr<uint8_t>(0);
        break;
    case 2:
        stInputParams.stImageData.stImageParams.eImageFormat =
RK_AVS_IMAGE_FORMAT_TYPE_YUVNV12;
        dataSize = stInputParams.stImageData.stImageParams.u32ImageStride *
stInputParams.stImageData.stImageParams.u32ImageHeight * 3 / 2;
        imageMat = cv::Mat(stInputParams.stImageData.stImageParams.u32ImageHeight
* 3 / 2, stInputParams.stImageData.stImageParams.u32ImageStride, CV_8UC1);
        pu8SrcImage = imageMat.ptr<uint8_t>(0);
        readBinParams(srcImagePath, (char*)pu8SrcImage, dataSize);
        break;
    default:
        std::cout << "RK_AVS_CALIB: ImageFormate error!" << std::endl;
        return 1;
    }
}
//输入输出路径

```

```

stInputParams.pcPointsImagePath = dstImagePath.c_str();
//读取图像
stInputParams.stImageData.pu8ImageData = pu8SrcImage;

//计时函数
#ifdef _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
    /*****角点提取*****/
    float stVerifyRms;
    int32_t status = rkAVS_findPoints(&stInputParams);
    if (status)
    {
        printf("rkAVS_calibFindPoints fail!\n");
    }
#ifdef _WIN32
    endTime = clock();
    timeAll = endTime - startTime;
    printf("rkAVS_calibFindPoints: %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
    return 0;
}

```

findPointsDemo为交代呢提取的demo,角点提取的结果会按照指定路径保存。

4. 标定失败及精度差的原因

4.1 拍图不符合要求

一下以错误拍摄的标定板效果为例，选自多个场景多种标定板规格下拍摄的问题图片。

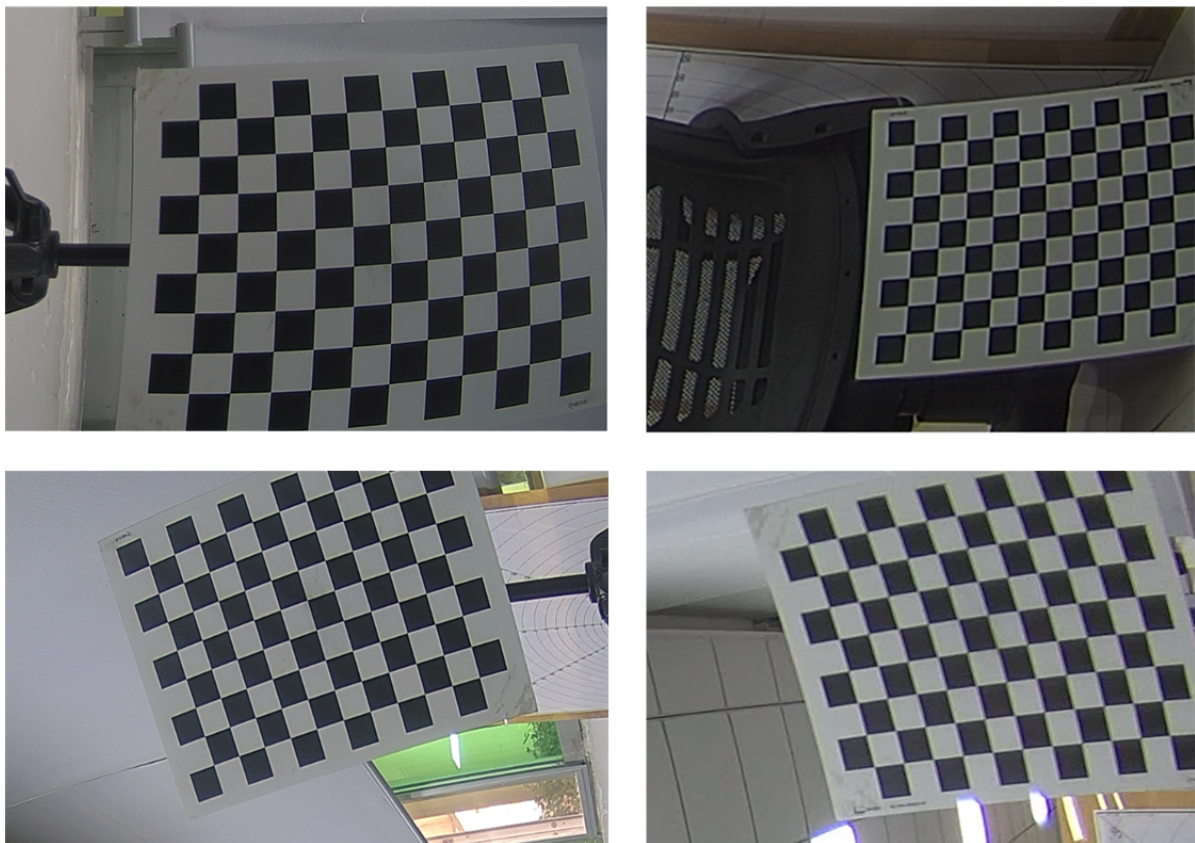
4.1.1 标定板反光

图4-1 反光



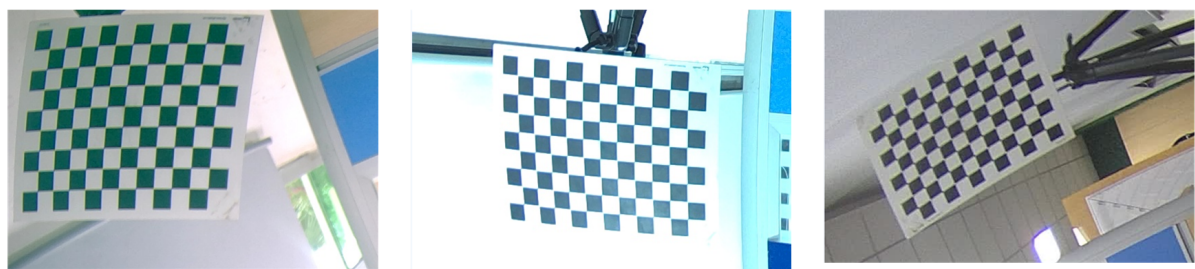
4.1.2 标定板未拍全

图4-2 棋盘格缺失



4.1.3 图像模糊

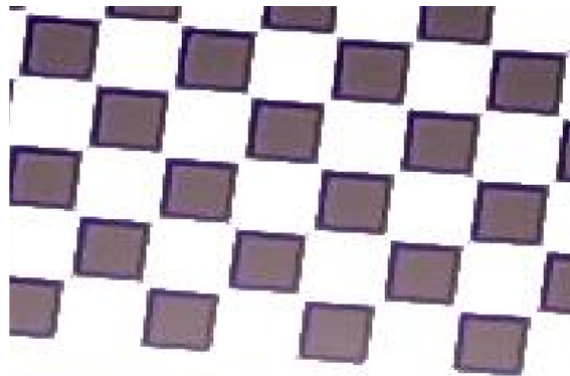
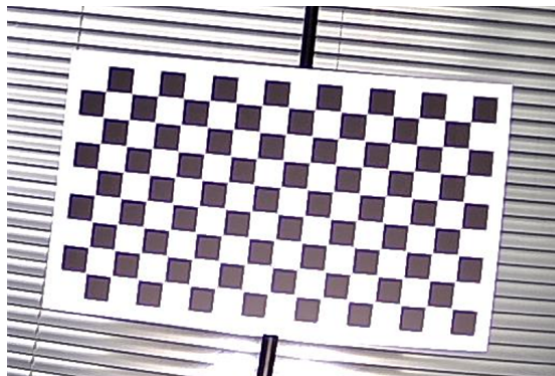
图4-3 棋盘格模糊



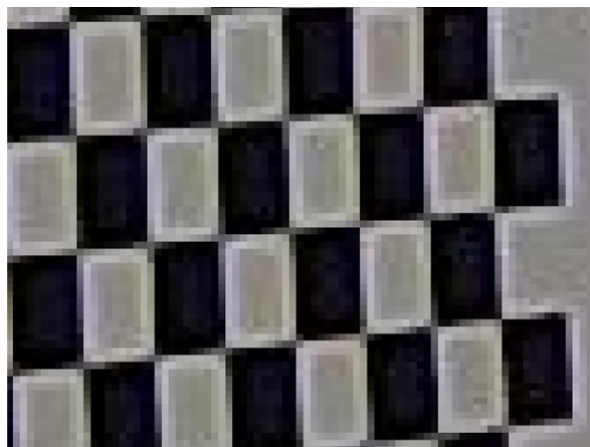
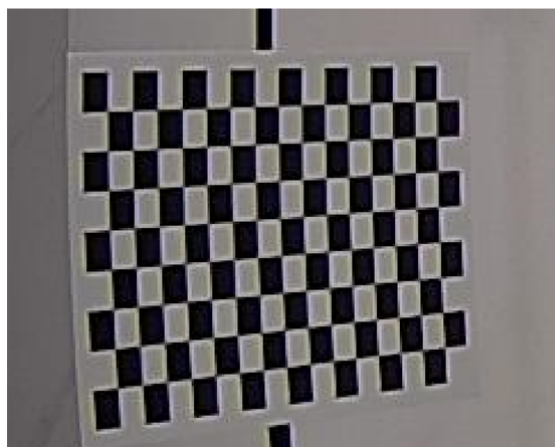
4.1.4 棋盘格不平整

张贴棋盘格不平整，存在鼓包等现象。

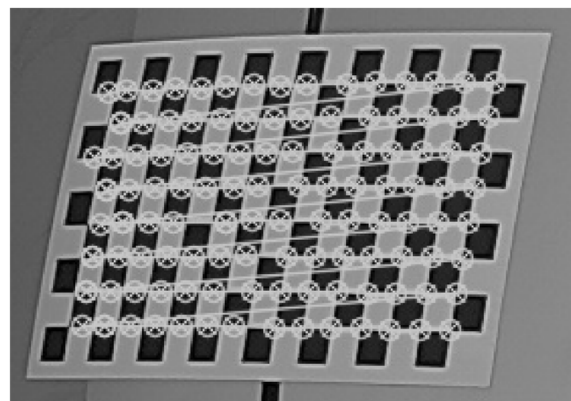
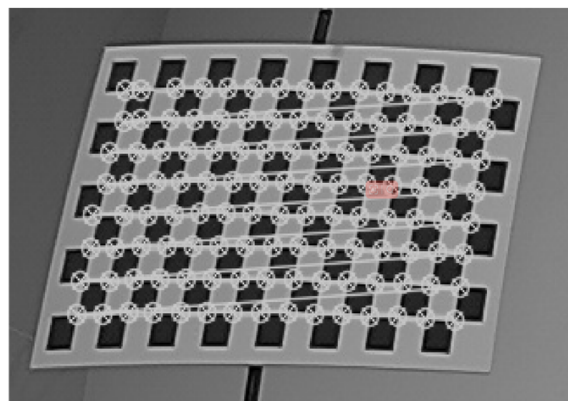
4.1.5 棋盘格过曝



4.1.6 光照不足



4.1.7 角点提取出错



4.2 参数配置错误

- 棋盘格角点个数配置出错

以图2-1为例子，该棋盘格横向有16个格子，纵向有9个格子，每个格子30mm，横向有15个内点，纵向有8个内点，配置棋盘格参数时，应按以下配置：

```
//棋盘格规格  
stInputParams.stBoardParams.u32BoardWidth = 15;  
stInputParams.stBoardParams.u32BoardHeight = 8;  
stInputParams.stBoardParams.u32SquareSize = 30;
```

- 图像尺寸配置出错

输入图像的宽、高、步长应配置正确。