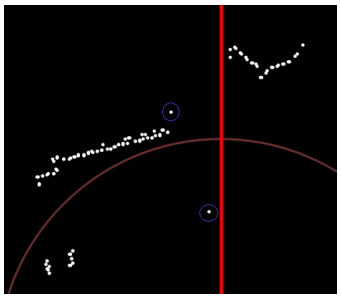
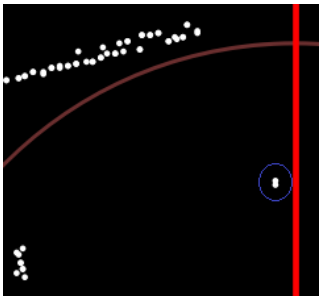
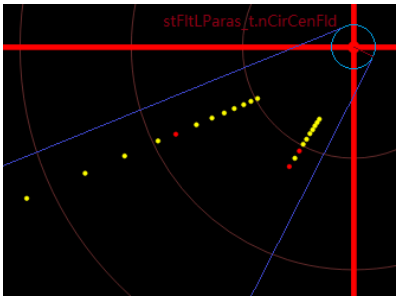
Toolkit V1.0.3R

一，接口简介

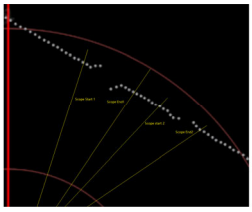
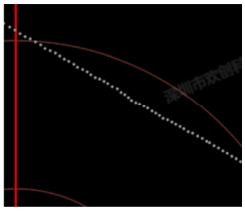
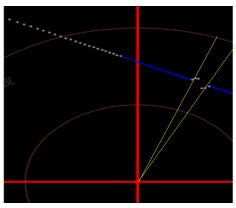
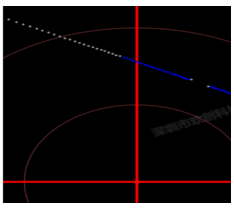
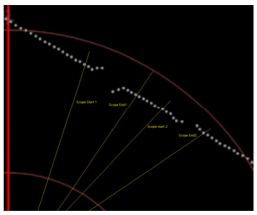
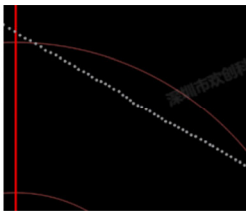
a，对 1 圈点云数据进行噪点滤除单孤点、双孤点、射线干扰点，单项可选开启/关闭

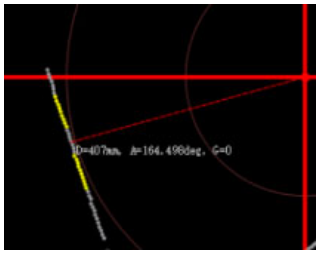
射线状干扰是指，强光、双激光互扰等造成的 line 状且趋势延长线与‘圆心域’内有园切的 Ray line 状干扰；

点状干扰是指，非聚类的孤立（单或 N）点；

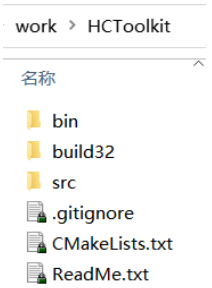
b，前柱 Shadow 集成了 Shadow 缺口清除、平滑、插补处理接口；

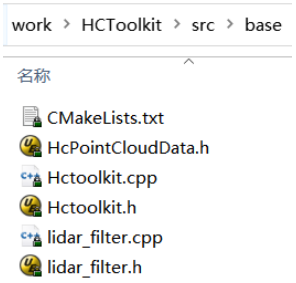







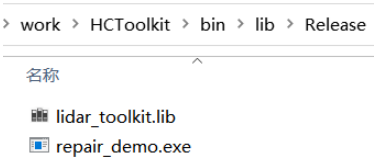
c，回充识别，基于特征码回充识别接口，基于亮度差的回充识别接口，
可设定识别功能区范围。



二，文件







三，编译 ，

Compile command (Need compiler which supports C++11)

1) Windows

mkdir build

cd build

cmake -G "Visual Studio 15 2017" ..

(Notice: please change the cmake command if you use other compiler)

2) Linux

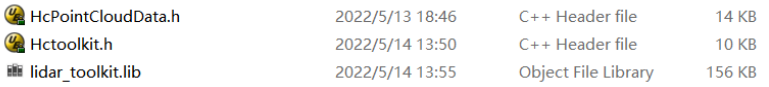
mkdir build

cd build

cmake ..

make

四，库、头文件



五，接口调用示例 repair_demo.cpp

```cpp
#include "base/Hctoolkit.h"

int main()
{
    int rtn = 0;

    string ss = GetVer();

    //If need Filter
    hcSDKFltInitialize();
    printf("hcSDKFltInitialize() complete\n");

    //To update lidar paras of model "X2B"
    stFltLidarCfg_t stLidarPara;
    bool bLowSpinSpeed = false;
    if(UpdateLidarPara("X2B", bLowSpinSpeed, stLidarPara))
    {
        printf("The stLidar Para with default value\n");
    }

    //The tsPtClouds of 1-circle Point Clouds to be processed
    std::vector<stPtCloud_t> tsPtClouds;
    printf("'tsPtClouds' size is  %d \n",tsPtClouds.size());

    //Can Overwrite stFltGblSetting or stFltLParas here,
    //or with the default value which defined with micro-Define in HcPointCloudData.h
    stFltGblSetting_t stFltGblSetting;
    stFltLParas_t stFltLParas;
    stFltGblSetting.fFacAdjHSpd = 3.0;
    stFltLParas.nActDist = 1000;

    //Call Filter function
    hotPixelFilter(tsPtClouds, stLidarPara, stFltGblSetting, stFltLParas);
    printf("Result Data in 'tsPtClouds' \n");

    //If need Filter, to Close modoule
    hcSDKFltUnInit();

    getchar();
    return 0;
} « end main »
```

六，接口函数

```cpp
//Noise Tools interface

/*****************************************************************************************************************************************/
//版本
/*****************************************************************************************************************************************/
char* GetVer();

/*****************************************************************************************************************************************/
//初始化
/*****************************************************************************************************************************************/
bool hcSDKFltInitialize();

/*****************************************************************************************************************************************/
//释放
/*****************************************************************************************************************************************/
bool hcSDKFltUnInit();

/*****************************************************************************************************************************************/
// 输入:
//       sLidarModel-eg.  "X2F"
//       bLowSpeed - true for Low Spin-Speed
// 输出:
//       stFltLidarCfg - Lidar Paras
// Return:
//       true: sLidarModel exisit in list,false: not in list
/*****************************************************************************************************************************************/
bool UpdateLidarPara(const char* sLidarModel, bool bLowSpeed, stFltLidarCfg_t &stFltLidarCfg);


//*****************************************************************************************************************************************/
//通用接口函数：作用域内,通过eFltType选择过滤器类型（普通、强光、通用场景）
//输入：
//       lstPointCloud - 1 圈点云数据
//       eFltType -  Filter 类型选择
//       stFltLidarCfg - Paras of 雷达
//       stFltGblSetting - Global 参数
//       stFltLParas - Line filter 参数
//输出：
//       lstPointCloud - 处理后的点云数据
//*****************************************************************************************************************************************/
void HotPixelFilterGblEx(ava_msg_lds_pack&     lstPointCloud,eFltType_t eFltType,const stFltLidarCfg_t &stFltLidarCfg,stFltGblSetting_t stFltGblSetting,stFltLParas_t stFltLParas);

//*****************************************************************************************************************************************/
//通用接口函数：作用域内,通过eFltType选择过滤器类型（普通、强光、通用场景）
//输入：
//       lstPointCloud - 1 圈点云数据
//       eFltType -  Filter 类型选择
//       stFltLidarCfg - Paras of 雷达
//       stFltGblSetting - Global 参数
//       stFltLParas - Line filter 参数
//输出：
//       lstPointCloud - 处理后的点云数据
//*****************************************************************************************************************************************/
void HotPixelFilterGbl(std::vector<stPtCloud_t> &lstPointCloud,eFltType_t eFltType,const stFltLidarCfg_t &stFltLidarCfg,stFltGblSetting_t stFltGblSetting,stFltLParas_t stFltLParas);
```

Pillar Shadow 接口

```
//*****************************************************************************************************************************/
//前柱遮挡区域的清除处理 - 在设定的距离、角度的扇形区域内
//输入 :
//       lstPointCloud - 1 圈点云数据
//       stPillarShadow - 要清除区域的起始角度、结束角度、区域的远距离，滤除平衡系数，滤除度设定：0/全点滤除 ，1/1孤点滤除，2/2孤点滤除，N/N孤点滤除
//       stFltLidarCfg - Paras of 雷达
//输出:
//       lstPointCloud - 处理后的点云数据
//*****************************************************************************************************************************/
void PillarShadowClear(lstPtCloud_t &lstPointCloud,const stFltLidarCfg_t &stFltLidarCfg,const stPillarShadow_t &stPillarShadow);


//*****************************************************************************************************************************/
//前柱缺口的补缺-在设定距离、角度的扇形区域内
//输入 :
//       lstPointCloud - 1 圈点云数据
//       stPillarShadow - 要清除区域的起始角度、结束角度、区域最远距离，修补迭代次数，Gap畸变最大阈值设定
//       stFltLidarCfg - Paras of 雷达
//输出:
//       lstPointCloud - 处理后的点云数据
//*****************************************************************************************************************************/
void PillarShadowSmooth(lstPtCloud_t &lstPointCloud,const stFltLidarCfg_t &stFltLidarCfg,stPillarShadow_t &stPillarShadow);


//*****************************************************************************************************************************/
//前柱缺口的补缺-在设定距离、角度的扇形区域内
//输入 :
//       lstPointCloud - 1 圈点云数据
//       stPillarShadow - 要清除区域的起始角度、结束角度、区域的远距离，修补迭代次数，Gap补缺角偏量（° 默认0）
//       stFltLidarCfg - Paras of 雷达
//输出:
//       lstPointCloud - 处理后的点云数据
//*****************************************************************************************************************************/
void PillarShadowRepair(lstPtCloud_t &lstPointCloud,const stFltLidarCfg_t &stFltLidarCfg,stPillarShadow_t &stPillarShadow);



//*****************************************************************************************************************************/
//前柱遮挡区域的清除处理 - 在设定的距离、角度的扇形区域内
//输入 :
//       lstPointCloud - 1 圈点云数据
//       stPillarShadow - 要清除区域的起始角度、结束角度、区域的远距离，滤除平衡系数，滤除度设定：0/全点滤除 ，1/1孤点滤除，2/2孤点滤除，N/N孤点滤除
//       stFltLidarCfg - Paras of 雷达
//输出:
//       lstPointCloud - 处理后的点云数据
//*****************************************************************************************************************************/
void PillarShadowClearEx(ava_msg_lds_pack& lstPointCloud,const stFltLidarCfg_t &stFltLidarCfg,const stPillarShadow_t &stPillarShadow);


//*****************************************************************************************************************************/
//前柱缺口的补缺-在设定距离、角度的扇形区域内
//输入 :
//       lstPointCloud - 1 圈点云数据
//       stPillarShadow - 要清除区域的起始角度、结束角度、区域最远距离，修补迭代次数，Gap畸变最大阈值设定
//       stFltLidarCfg - Paras of 雷达
//输出:
//       lstPointCloud - 处理后的点云数据
//*****************************************************************************************************************************/
void PillarShadowSmoothEx(ava_msg_lds_pack& lstPointCloud,const stFltLidarCfg_t &stFltLidarCfg,stPillarShadow_t &stPillarShadow);


//*****************************************************************************************************************************/
//前柱缺口的补缺-在设定距离、角度的扇形区域内
//输入 :
//       lstPointCloud - 1 圈点云数据
//       stPillarShadow - 要清除区域的起始角度、结束角度、区域的远距离，修补迭代次数，Gap补缺角偏量（° 默认0）
//       stFltLidarCfg - Paras of 雷达
//输出:
//       lstPointCloud - 处理后的点云数据
//*****************************************************************************************************************************/
void PillarShadowRepairEx(ava_msg_lds_pack& lstPointCloud,const stFltLidarCfg_t &stFltLidarCfg,stPillarShadow_t &stPillarShadow);
```

回充识别接口

```
//*****************************************************************************************************************************/
//通用接口函数：定位回充基站
//输入 :
//       lstPointCloud - 1 圈点云数据
//       eRechargeType -  定位类型
//       stFltLidarCfg -  雷达参数
//       stReChargeSetting - 全局配置参数
//       stReChargeSetting - 回充设定参数
//输出:
//       stRechargeDir - 定位参数
//*****************************************************************************************************************************/
void RechargeLocateGbl(lstPtCloud_t &lstPointCloud,const eRechargeType_t &eRechargeType,const stFltLidarCfg_t &stLidarCfg, stRechargeSetting_t &stReChargeSetting,stRechargeDir_t &stRechargeDir);

/*****************************************************************************************************************************/
// 初始化点云回充特征码参数
// 输入:
//       na1,nb1,na2,nb2,  -   雷达属性包上传的特征码系数值
//       nSpecCalibDist1 -  指定值(有默认值),对应于'特征码系数值'的距离参数-1
//       nSpecCalibDist2 -  指定值(有默认值),对应于'特征码系数值'的距离参数-2
//       nSpecValue      -  回充特征码值(默认255)
// 输出:
//       m_stSpecCodeParas   -  初始化后的回充特征参数
/*****************************************************************************************************************************/
void InitSpecCodeParas(const int &na1,const int &nb1,const int &na2,const int &nb2,const int &nSpecValue,
          const int &nSpecCalibDist1,const int &nSpecCalibDist2,const std::vector<stSpecGrayDifCalIb_t> &lstGrayDifCalib,const int &nBarGraySecBrkDif,const int &nBarGrayFltDif,const int &nSpecGrayMin,
          const int &nSpecGrayMax);
```

回充接口例程

```
m_stRechareDirect.angle_disp = 0;
m_stRechareDirect.dist = 0;
if(g_sSysPara.nShadowWorkType >= PILLAR_GAP_NOTHING && g_sSysPara.nRechargeWorkType < RECHARGE_NOTHING)
{
    stRechargeDir_t  stRechargeDir;
    std::vector<stSpecGrayDifCalib_t> lstGrayCalib;
    if(g_sSysPara.nRechargeWorkType == RECHARGE_SPEC_CODE)  //特征码回充调用
    {
        std::vector<UINT16> lstRch;
        getSDKRechargeValue(lstRch);
        if(lstRch.size())
        {
            InitSpecCodeParas(lstRch[0],lstRch[1],lstRch[2],lstRch[3],g_sSysPara.stRechSpecCodeParas.nSpecValue,
                              g_sSysPara.stRechSpecCodeParas.nSpecCalibDist1,g_sSysPara.stRechSpecCodeParas.nSpecCalibDist2,
                              lstGrayCalib.g_sSysPara.stRechSpecCodeParas.nBarGraySecBrkDif,g_sSysPara.stRechSpecCodeParas.nBarGrayFltDif,
                              g_sSysPara.stRechSpecCodeParas.nSpecGrayMin,g_sSysPara.stRechSpecCodeParas.nSpecGrayMax);
            RechargeLocateGbl(tsPtCloudData,(eRechargeType_t)g_sSysPara.nRechargeWorkType,stFltLidarCfg,g_sSysPara.stRechargeSetting,stRechargeDir);
            m_stRechareDirect.angle_disp =  fmod(stRechargeDir.dAngleTo + g_sSysPara.sProject.fPointCloudAng + m_fAngOffset,360.0);
            m_stRechareDirect.dist = stRechargeDir.nDistTo;
        }
    }
    else    //灰度差回充调用
    {
        std::vector<stSpecGrayDifCalib_t> lstGrayCalib;
        getSDKRechargeGrayDifCalib(lstGrayCalib);
        InitSpecCodeParas(0,0,0,0,g_sSysPara.stRechSpecCodeParas.nSpecValue,
                          g_sSysPara.stRechSpecCodeParas.nSpecCalibDist1,g_sSysPara.stRechSpecCodeParas.nSpecCalibDist2,
                          lstGrayCalib.size()? lstGrayCalib:g_sSysPara.stRechSpecCodeParas.lstSpecGrayDifCalib,g_sSysPara.stRechSpecCodeParas.nBarGraySecBrkDif,g_sSysPara.stRechSpecCodeParas.nBarGrayFltDif,
                          g_sSysPara.stRechSpecCodeParas.nSpecGrayMin,g_sSysPara.stRechSpecCodeParas.nSpecGrayMax);
        RechargeLocateGbl(tsPtCloudData,(eRechargeType_t)g_sSysPara.nRechargeWorkType,stFltLidarCfg,g_sSysPara.stRechargeSetting,stRechargeDir);

        m_stRechareDirect.angle = stRechargeDir.dAngleTo;
        m_stRechareDirect.angle_disp =  fmod(stRechargeDir.dAngleTo + g_sSysPara.sProject.fPointCloudAng + m_fAngOffset,360.0);
        m_stRechareDirect.dist = stRechargeDir.nDistTo;
    }
} « end if g_sSysPara.nShadowWor... »
```

七，参数定义

1，输入点云数据类型

dAngle，u16Dist，u16Gray，bGrayTwoByte，dAngleRaw 是 toolkit 使用项，“其它“后面为选配项

```
//用于定义 点云 类型，Toolkit 用于处理1圈点云数据
typedef struct _stPtCloud
{
                                        //Toolkit 涉及变量
    bool        bValid;                 // true:有效点, false:非有效点
    double      dAngle;                 // 点云角度 °
    UINT16      u16Dist;                // 点云距离,mm
    UINT16      u16Gray;                // 亮度
    UINT16      u16Speed;               // 雷达转速, RPM

    bool        bGrayTwoByte;           // true：u16Gray存储2字节亮度信息,false u16Gray 存储1字节亮度信息
    double      dAngleRaw;              // 矫正前的初始点云角度 °（前柱补缺模块中使用的变量）
    UINT8       u8Stable;               // 保留位
    UINT8       u8Row;                  // 行
    UINT8       u8Col;                  // 列

                                        // 其它 可选配
    double      dAngleDisp;             // 点云显示工具用变量，无用到
    UINT16      u16DistRaw;             // 初始点云距离,mm
    UINT64      u64TimeStampMs;         // 时间戳 ,ms
    float       fTemperature;           // 温度

    _stPtCloud() :
        bValid(true),
        dAngle(0.),
        dAngleRaw(0.),
        u16Dist(0),
        u16Gray(0),
        bGrayTwoByte(false),
        u8Stable(0),
        u8Row(0),
        u8Col(0),

        dAngleDisp(0.),
        u16DistRaw(0),
        u16Speed(0),
        u64TimeStampMs(0),
        fTemperature(0)
    {}
} « end _stPtCloud » stPtCloud_t;
typedef std::vector<stPtCloud_t> lstPtCloud_t;
```

2，Toolkit 内部使用类型定义

```
typedef struct _FltCloudD               //Flt Cloud Data
{
    double      x;                      //X 坐标 x = dist * cos(angle_rad_cur);
    double      y;                      //Y 坐标 y = dist * sin(angle_rad_cur);
    UINT16      u16Dist;                //距离
    double      dAngle;                 //原始角度 (°)
    bool        bValidIs;               //true:有效点 false:无效点
    int         nfound;                 //程序动态标记

    _FltCloudD():
        x(65535),
        y(65535),
        u16Dist(65535),
        dAngle(0),
        bValidIs(1),
        nfound(0)
    {}
}stFltCloudD_t;
```

3，Lidar 类型定义

```
#define  DIST_INNER_CIRCLE      140      //mm Blind area size; eg. D2?:140, X2? 120, X1?100
#define  LIDAR_FPS              3000     //帧率/second
#define  LIDAR_RPM              360      //转速/minute
#define  H_L_SPD_THRES          210      //Threshold of High/Low Spin-speed,design value;Eg. X2MF(180/360 L/H RMS) : 270,  X2F(360 RMS):270,D2S(300):210

typedef struct _FltLidarCfg             //Lidar Paras
{
    unsigned int nBlindDist;            //盲区内被清除的距离 mm
    unsigned int nFPS;                  //帧率/second
    unsigned int nSpinSpeed;            //转速/minute: 设计指标，高低转速切换时，需更新此值
    int          nLHSpdThres;           //Threshold of High/Low Spin-speed,design value;Eg. X2MF(180/360 L/H RMS) : 270,  X2F(360 RMS):270,D2S(300):210

    _FltLidarCfg():
        nFPS(LIDAR_FPS),
        nSpinSpeed(LIDAR_RPM),
        nBlindDist(DIST_INNER_CIRCLE),
        nLHSpdThres(H_L_SPD_THRES)
    {}
}stFltLidarCfg_t;
```

4，射线滤除类型定义

```
#define  ORG_REGION_L           65       //mm, Ray valid area,Better to opt.till now need not;
#define  DIST_CLEAR_MAX_L       600      //mm, Line Clear Max,Better to opt
#define  LINAR_OFST_MAX_L       3        //Max offset of Spot to the ray
#define  LINAR_CNT_L            3        //Min Spots of Ray

typedef struct _FltLParas               //L filter paras
{
    unsigned int nActDist;              //mm,Ray-line 滤除有效范围，如600mm
    unsigned int nCirCenFld;            //mm,射线到原点距离阈值，如65mm
    unsigned int nPtToLineMin;          //mm,同一直线点的距离阈值，如3mm
    unsigned int nPtOfLineMin;          //点->线最小点数阈值

    _FltLParas()
    {
        nActDist =   DIST_CLEAR_MAX_L;
        nCirCenFld = ORG_REGION_L;
        nPtToLineMin = LINAR_OFST_MAX_L;
        nPtOfLineMin = LINAR_CNT_L;
    }
}stFltLParas_t;
```

4，滤除范围、均衡性类型定义

```
#define  SHARK_GRAY_THRES       50       //bGrayTwoByte true:400,false:50
#define  HALOGEN_GRAY_THRES     7000     //Feature of Stronglight, need to adj., eg. bGrayTwoByte true:7000,false:255
#define  ANG_COMP_OFST          50       //Angle offset value
#define  DIST_OUTER_CIRCLE      1000     //mm Range-limit,may Adjust as required

#define  FAC_HSPD_ADJ           3.0      //balance factor of high RPM, is to spot which is filtered out or left
#define  FAC_LSPD_ADJ           3.0      //balance factor of low RPM,is to spot which is filtered out or left

typedef struct _FltGblSetting
{
    float        fFacAdjHSpd;           //杂点滤除平衡系数-高转速
    float        fFacAdjLSpd;           //杂点滤除平衡系数-低转速
    bool         bFltRayLOn;            //true: RayLine Flt on, false:Off
    bool         bFlt1DotOn;            //true: 1 Dot Flt on, false:Off
    bool         bFlt2DotOn;            //true: 2 Dot Flt on, false:Off, active in strong-light region
    int          nFltNDotOn;            //N>2: N Dots Flt on, 0: Off, active in strong-light region
    unsigned int nActDist;              //整体作用域（单位：mm）
    double       dActScopeSt;           //(0-360)Action is limited in the region of st-end,if dActScopeSt == dActScopeEnd mean the angle-limitation is invalid
    double       dActScopeEnd;          //(0-360)Action is limited in the region of st-end,if dActScopeSt == dActScopeEnd mean the angle-limitation is invalid
    bool         bScopeSmallIs;         //true: acting in small region between dActScopeSt and dActScopeEnd, false: acting in big region between dActScopeSt and dActScopeEnd
    bool         bPtActiveSt;           //true or false, Valid Point Flag Setting
    unsigned int nHalogenGrayThres;     //7000+,Feature of Stronglight, need to adj., eg. bGrayTwoByte true:7000,false:255
    unsigned int nSharkGrayThres;       //bGrayTwoByte true:400,false:50
    unsigned int nAngExtent;            //(+ Center -),Range (°)

    _FltGblSetting()
    {
        fFacAdjHSpd = FAC_HSPD_ADJ;
        fFacAdjLSpd = FAC_LSPD_ADJ;
        bFltRayLOn = true;
        bFlt1DotOn = true;
        bFlt2DotOn = true;
        nFltNDotOn = 0;
        nActDist = DIST_OUTER_CIRCLE;
        dActScopeSt = 0.0;
        dActScopeEnd  = 0.0;
        bScopeSmallIs = true;
        bPtActiveSt = true;
        nHalogenGrayThres = HALOGEN_GRAY_THRES;
        nSharkGrayThres = SHARK_GRAY_THRES;
        nAngExtent = ANG_COMP_OFST;
    }
} « end _FltGblSetting » stFltGblSetting_t;
```

5，前柱 Shadow 类型定义

```
#define PILLAR1_SHADOW_ANG_ST           0
#define PILLAR1_SHADOW_ANG_END          0
#define PILLAR1_SHADOW_ANG_ST_2         0
#define PILLAR1_SHADOW_ANG_END_2        0

#define PILLAR2_SHADOW_ANG_ST           0
#define PILLAR2_SHADOW_ANG_END          0
#define PILLAR2_SHADOW_ANG_ST_2         0
#define PILLAR2_SHADOW_ANG_END_2        0

#define PILLAR3_SHADOW_ANG_ST           0
#define PILLAR3_SHADOW_ANG_END          0
#define PILLAR3_SHADOW_ANG_ST_2         0
#define PILLAR3_SHADOW_ANG_END_2        0

#define PILLAR_CLEAR_NODE_PT_NUM        0
#define PILLAR_SMOOTH_NODE_PT_NUM       5

#define PILLAR_CLEAR_SHADOW_FAC_ADJ     1.0
#define PILLAR_REP_DIST_THRES           150
#define PILLAR_REP_ITERATOR_NUM         5
#define PILLAR_REP_PILLAR_DIST          50
#define PILLAR_REP_ANG_OFFSET           0
```

```
typedef struct _PillarShadowParas          //前柱遮挡区域的处理参数
{
    unsigned int nActDist;                //mm,Pillar shadow 处理的有效范围，如60000mm

    double       d1ShadowScopeSt;          //柱1，如无，设为0; Shadow region Gap-1 start angle (°); 0:功能关闭
    double       d1ShadowScopeEnd;         //柱1，如无，设为0; Shadow region Gap-1 end angle (°);   0:功能关闭
    double       d1ShadowScopeSt2;         //柱1，如无，设为0
    double       d1ShadowScopeEnd2;        //柱1，如无，设为0

    double       d2ShadowScopeSt;          //柱2，如无，设为0
    double       d2ShadowScopeEnd;         //柱2，如无，设为0
    double       d2ShadowScopeSt2;         //柱2，如无，设为0
    double       d2ShadowScopeEnd2;        //柱2，如无，设为0

    double       d3ShadowScopeSt;          //柱3，如无，设为0
    double       d3ShadowScopeEnd;         //柱3，如无，设为0
    double       d3ShadowScopeSt2;         //柱3，如无，设为0
    double       d3ShadowScopeEnd2;        //柱3，如无，设为0


    int          nRepIteratorNum;         //修补迭代次数
    int          nRepGapDistThr;          //Gap畸变阈值设定 mm
    int          nRepPillarDist;          //前柱距离范围阈值mm
    double       dRepAngAdj;              //Gap补缺角度偏调量(°)(非0时,此值起效; 0时,若dAngleRaw非0,则使用dAngleRaw进行纠偏)

    int          nClearNodePtNum;         //滤除点数设定：0/全点滤除 , 1/点孤点滤除, 2/2孤点滤除, N/N孤点滤除
    int          nSmoothNodePtNum;        //平滑Node 点数：3,5,7
    float        fClearFacAdj;            //杂点滤除平衡系数调整


    _PillarShadowParas()
    {
        nActDist =   DIST_PILLAR_SHADOW;
        d1ShadowScopeSt = PILLAR1_SHADOW_ANG_ST;
        d1ShadowScopeEnd = PILLAR1_SHADOW_ANG_END;
        d1ShadowScopeSt2 = PILLAR1_SHADOW_ANG_ST_2;
        d1ShadowScopeEnd2 = PILLAR1_SHADOW_ANG_END_2;

        d2ShadowScopeSt = PILLAR2_SHADOW_ANG_ST;
        d2ShadowScopeEnd = PILLAR2_SHADOW_ANG_END;
        d2ShadowScopeSt2 = PILLAR2_SHADOW_ANG_ST_2;
        d2ShadowScopeEnd2 = PILLAR2_SHADOW_ANG_END_2;

        d3ShadowScopeSt = PILLAR3_SHADOW_ANG_ST;
        d3ShadowScopeEnd = PILLAR3_SHADOW_ANG_END;
        d3ShadowScopeSt2 = PILLAR3_SHADOW_ANG_ST_2;
        d3ShadowScopeEnd2 = PILLAR3_SHADOW_ANG_END_2;

        nClearNodePtNum = PILLAR_CLEAR_NODE_PT_NUM;
        nSmoothNodePtNum = PILLAR_SMOOTH_NODE_PT_NUM;
        fClearFacAdj = PILLAR_CLEAR_SHADOW_FAC_ADJ;
        nRepIteratorNum = PILLAR_REP_ITERATOR_NUM;
        nRepGapDistThr = PILLAR_REP_DIST_THRES;
        nRepPillarDist = PILLAR_REP_PILLAR_DIST;
        dRepAngAdj = PILLAR_REP_ANG_OFFSET;
    } « end _PillarShadowParas »
} « end _PillarShadowParas » stPillarShadow_t;
```

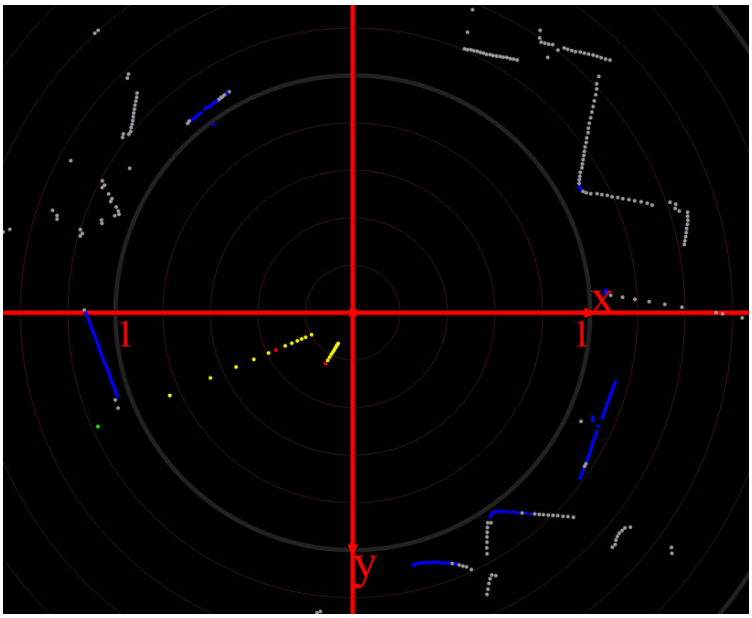
6，回充识别类型定义

```
typedef struct _SpecGrayDifCalib       //回充特征定义
{
    UINT16        nCalibDist;              //标定距离 mm
    UINT16        nGrayDifValue;           //标定差值
}stSpecGrayDifCalib_t;


#define RECHAR_CALI_DIST_1           250     //特征码距离参数 mm
#define RECHAR_CALI_DIST_2           650     //特征码距离参数 mm
#define RECHAR_SPEC_CODE             255     //特征码距离参数 mm
#define RECHAR_SPEC_LIGHT_MIN        0       //特征靶亮度范围设定
#define RECHAR_SPEC_LIGHT_MAX        255     //特征靶亮度范围设定
#define RECHAR_SPECBAR_GRAY_DIF      10000   //特征靶亮度差值大小
#define RECHAR_SPECBAR_GRAY_SEC_DIF  100     //特征靶亮度差值Secion-break阈值
#define RECHAR_SPECBAR_GRAY_FLT_DIF  10000   //特征靶亮度差值滤除阈值大小（边缘）


typedef struct _SpecCodeParas        //回充特征定义
{
    double       dA1;                    //特征码参数
    int          nB1;                    //特征码参数
    double       dA2;                    //特征码参数
    int          nB2;                    //特征码参数
    UINT16       nSpecCalibDist1;        //特征码距离参数 mm
    UINT16       nSpecCalibDist2;        //特征码距离参数 mm
    UINT16       nSpecValue;             //特征码默认值设定

    UINT16       nSpecGrayMin;           //特征靶亮度范围设定
    UINT16       nSpecGrayMax;           //特征靶亮度范围设定
    //UINT16       nBarGrayDif;            //特征靶亮度差值阈值
    UINT16       nBarGraySecBrkDif;      //特征靶亮度差值Secion-break阈值
    UINT16       nBarGrayFltDif;         //特征靶亮度差值滤除阈值大小（边缘）

    std::vector<stSpecGrayDifCalib_t> lstSpecGrayDifCalib;  //特征靶亮度差值阈值

    _SpecCodeParas():
    dA1(0),
    nB1(0),
    dA2(0),
    nB2(0),
    nSpecCalibDist1(RECHAR_CALI_DIST_1),
    nSpecCalibDist2(RECHAR_CALI_DIST_2),
    nSpecValue(RECHAR_SPEC_CODE),
    nSpecGrayMin(RECHAR_SPEC_LIGHT_MIN),
    nSpecGrayMax(RECHAR_SPEC_LIGHT_MAX),
    nBarGraySecBrkDif(RECHAR_SPECBAR_GRAY_SEC_DIF),
    nBarGrayFltDif(RECHAR_SPECBAR_GRAY_FLT_DIF)
    {
        //nBarGrayDif(RECHAR_SPECBAR_GRAY_DIF);
        //stSpecGrayDifCalib_t stSpecGrayDifCalib={3000,RECHAR_SPECBAR_GRAY_DIF};
        //lstSpecGrayDifCalib.push_back(stSpecGrayDifCalib);
    }
} « end _SpecCodeParas » stSpecCodeParas_t;
```

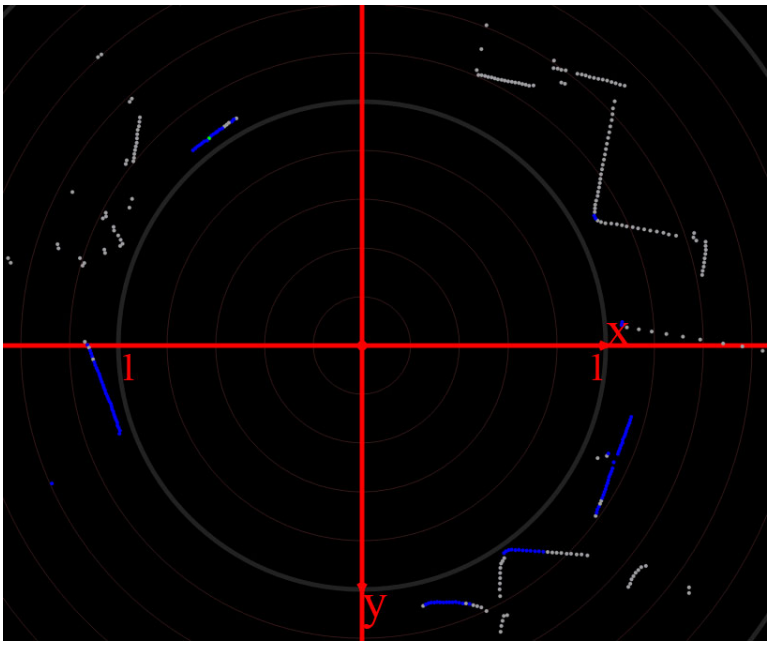
```
#define RECHAR_FAR_THRES            1200  //远/近距离阈值
#define RECHAR_SPECBAR_LEN          500   //特征靶材宽度 mm
#define RECHAR_SPECBAR_BIAS         0     //特征靶材的宽度偏差 mm
#define RECHAR_SPECBAR_INTERVAL     1500  //特征靶间隔长度 mm
#define RECHAR_SPECINTERVAL_BIAS    100   //特征靶间隔长度偏差 mm
#define RECHAR_MAX_DIST             3000  //特征靶识别最远距离 mm
#define RECHAR_CREDIT_TRRES         1     //1-N 次

typedef struct _RechargeSetting      //回充参数定义
{
    UINT16       nCreditDThres;          //Crediting Degree setting
    UINT16       nDetDistMax;            //特征靶识别最远距离 mm
    double       dActScopeSt;            //识别功能区角度范围设定,<0, 0>即dActScopeSt==dActScopeEnd 则360°范围内识别
    double       dActScopeEnd;           //识别功能区角度范围设定,<0, 0>即dActScopeSt==dActScopeEnd 则360°范围内识别
    bool         bScopeSmallIs;          //true: <dActScopeSt, ddActScopeEnd>小角度范围内识别; false: <dActScopeSt, ddActScopeEnd>大角度范围内识别
    bool         bSlicleIs;              //特征码是否存在断层情况
    UINT16       nFarThr;                //远/近距离阈值

    UINT16       nSpecBarLen;            //特征靶材宽度 mm
    short        nSpecBarBias;           //特征靶材的宽度偏差mm
    UINT16       nBarInterval;           //特征靶间隔长度 mm
    short        nBarIntervalBias;       //特征靶间隔长度偏差 mm

    _RechargeSetting()
    {
        nCreditDThres = RECHAR_CREDIT_TRRES;
        dActScopeSt = 0;
        dActScopeEnd = 0;
        bScopeSmallIs = true;
        bSlicleIs = false;
        nFarThr = RECHAR_FAR_THRES;
        nSpecBarLen = RECHAR_SPECBAR_LEN;
        nSpecBarBias = RECHAR_SPECBAR_BIAS;

        nBarInterval = RECHAR_SPECBAR_INTERVAL;
        nBarIntervalBias = RECHAR_SPECINTERVAL_BIAS;
        nDetDistMax = RECHAR_MAX_DIST;
    }
} « end _RechargeSetting » stRechargeSetting_t;

typedef struct _RechargeCldD         //基站点云数据类型定义
{
    double          x;                  //X 坐标 x = dist * cos(angle_rad_cur);
    double          y;                  //Y 坐标 y = dist * sin(angle_rad_cur);
    unsigned short  u16Dist;            //距离
    double          dAngle;             //角度 (°) , 原始角度 or 显示角度 or 纠偏后角度, 需具一致性;
    unsigned short  u16Gray;            //灰度
    bool            bValidIs;           //true:有效点 false:无效点

    int             nIdx;               //点云中的序数
    UINT16          u16SpecCode;        //回充特征码
    bool            bFound;             //程序动态标记

    _RechargeCldD():
    x(0),
    y(0),
    u16Dist(0),
    dAngle(0),
    bValidIs(1),
    nIdx(0),
    u16SpecCode(0),
    bFound(false)
    {}
} « end _RechargeCldD » stRechargeCldD_t;

typedef struct _RecharCldSection     //基站数据类型
{
    std::vector <stRechargeCldD_t> lstRechgCldD;
    int             nStIdx;
    int             nEndIdx;
    _RecharCldSection():
    nStIdx(0),
    nEndIdx(-1)
    {}
}stRecharCldSection_t;

typedef struct _RechargeDirection  //基站方向
{
    double    dAngleTo;
    double    dDistTo;
    _RechargeDirection():
    dAngleTo(0),
    dDistTo(0)
    {}
}stRechargeDir_t;
```

八，效果示例（起效范围设定 1 米，0°- 360°）

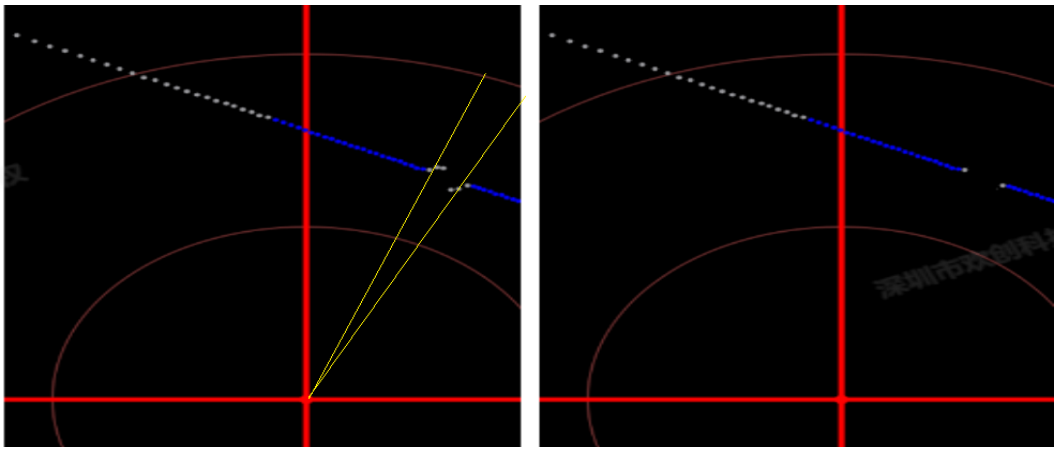
调用前



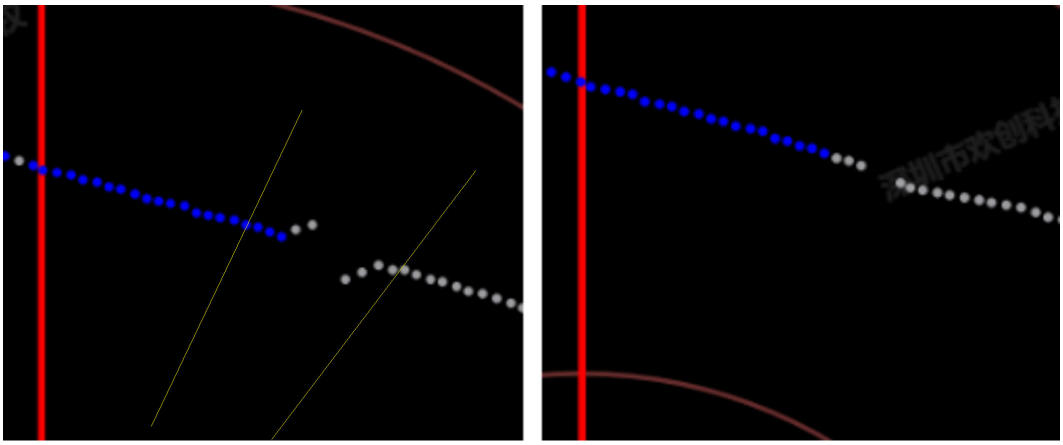
调用后



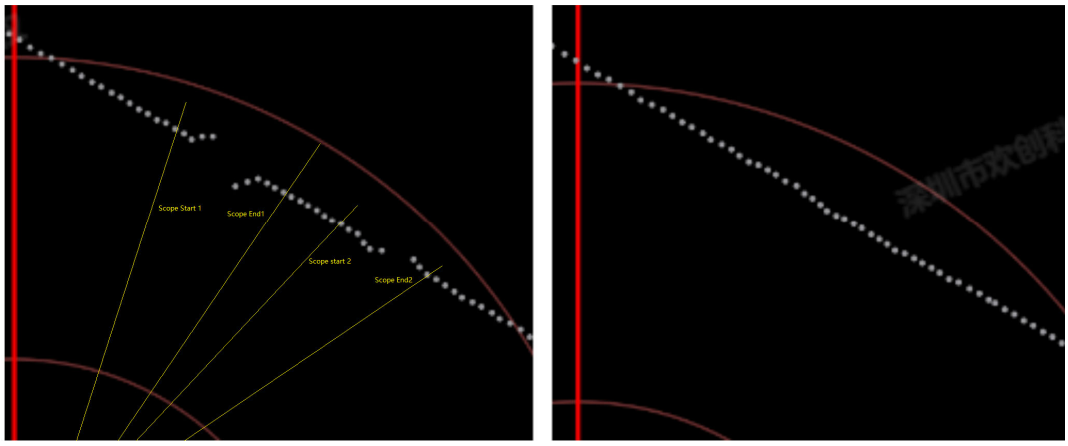
a，Shadow Gap 滤除处理



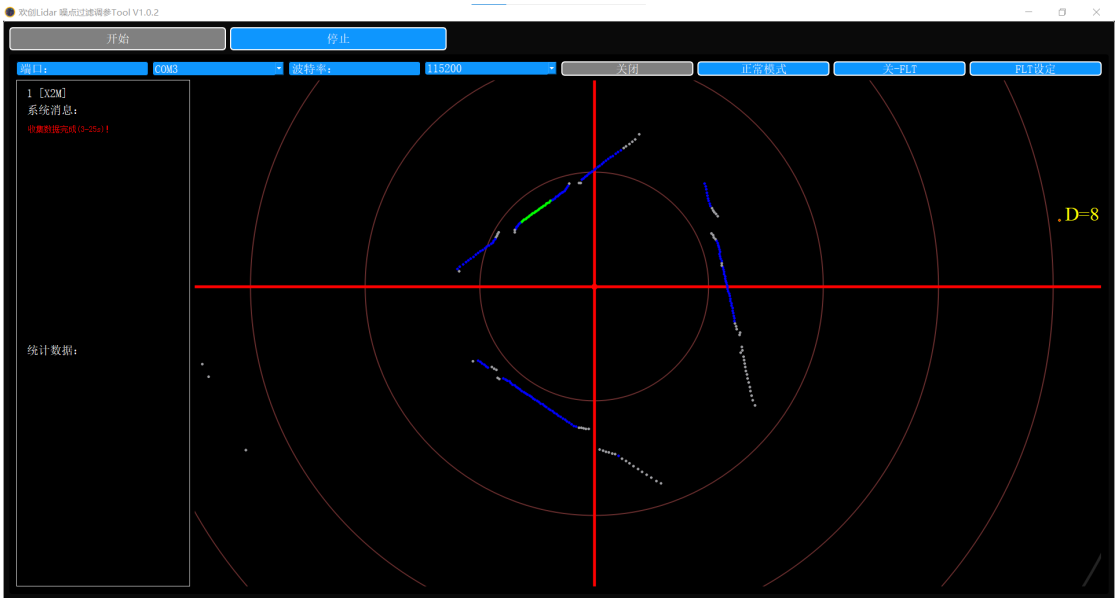
b，Shadow 平滑处理



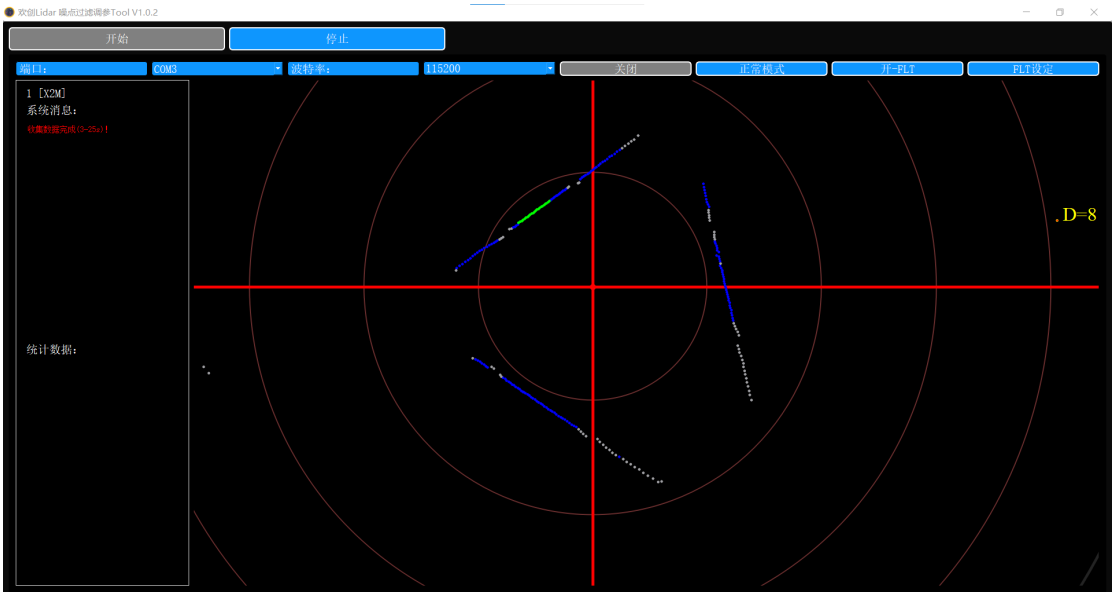
c，Shadow Gap 补缺处理



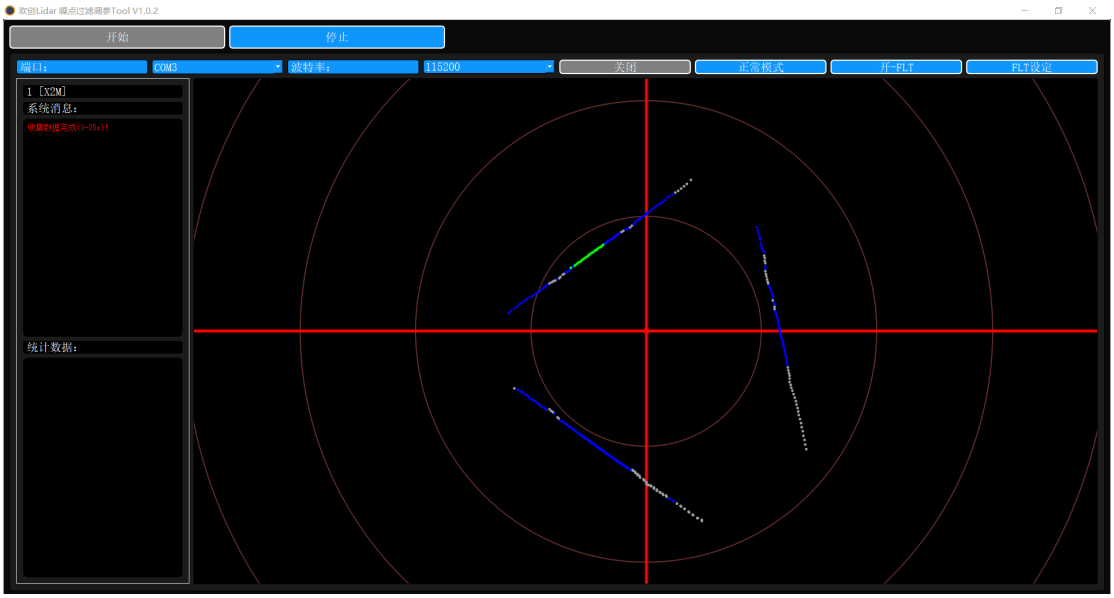
前柱缺口处理关



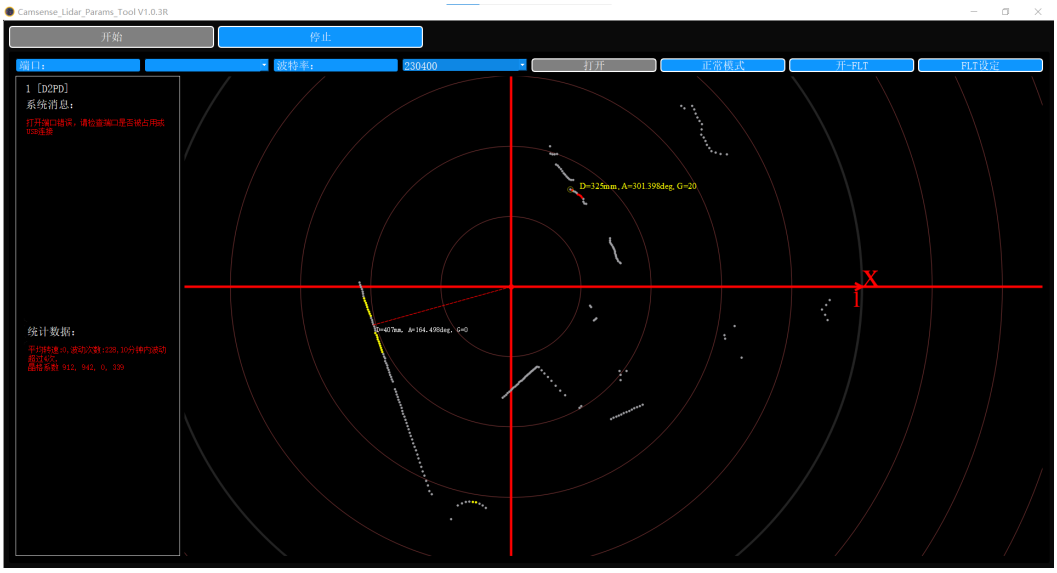
前柱缺口平滑开



前柱补缺开


欢创Lidar 噪点过滤调参Tool V1.0.2
开始
停止
端口：
COM3
波特率：
115200
关闭
正常模式
开-FLT
FLT设定
1 [X2M]
系统消息：
统计数据：

回充识别


Camsense_Lidar_Params_Tool V1.0.3R
开始
停止
端口：
波特率：
230400
打开
正常模式
开-FLT
FLT设定
1 [D2PD]
系统消息：
统计数据：
D=325mm, A=301.398deg, G=20
X