基于交叉结构光视觉传感器的智能焊缝识别系统（课题研究方向）

项目背景

该项目与本人在读期间的研究方向，该项目与上海某激光焊接技术有限公司合作，旨在开发一种用于智能焊接的新型激光视觉传感器，来解决实际工业焊接场景下的焊接初始点无法识别，适应焊缝类型单一、算法鲁棒性差等难点。

作为研究生课题方向，项目开发期间，大部分时间花在了研究和提出可行性的创新算法上，所以项目的持续时间较久。

主要工作：

- 通过自主设计的激光视觉传感器（单相机+交叉结构光发生器），采集工业场景中的焊缝工件图像。
- 提出一种创新的图像算法，处理交叉结构光映射图像，自动识别焊缝参数（类型、初始点坐标、宽度、倾斜角度等）。
- 用 QT 开发了初步的人机交互界面，包含基本的相机操作按钮（查找设备、打开关闭、图像采集、图像显示区、以及图像处理结果和处理数据），实现了基本的应用功能。
- 将循环读取相机内部缓存、图像处理这些耗时任务放在新线程中处理，提高程序的并发性和响应效率。

个人收获

技术方面：

涉及技术：C++、Python、QT、Opencv、图像处理、坐标变换、相机标定等

项目管理与沟通方面：

为了保证一定的项目进度，所以我一般会在时间表上做一个大致的规划，按照计划去推进进度。在发生一些意外情况的时候，及时进行沟通协调，确保项目的顺利开展。

代码类

LaserSensor 主窗口类

该类主要负责构建主窗口的相关构件，实现相机调用和图像采集，并调用相应图像处理算法进行识别。

```
#ifndef LASERVISIONSENSOR_H
#define LASERVISIONSENSOR_H

#include "ui_LaserVisionSensor.h"

#include <QtWidgets/QWidget>
#include <QMessageBox>
#include <QCloseEvent>
#include <QSettings>
#include <QDebug>
#include <QWidget>
#include <QValidator>
#include <qdatetime.h>

#include <opencv2\opencv.hpp>
#include <fstream>
#include <iostream>
#include <ctime>
#include <sstream>

#include "MvCamera.h"
#include "MyCameraThread.h"

#include "T_Common.h"
#include "T_Lap.h"
//#include "T_SingleBevel.h"
//#include "T_Square.h"
//#include "T_V.h"

#define TRIGGER_SOURCE 7 // 触发源（默认软触发）
#define EXPOSURE_TIME   80000 // 曝光时间 us
#define FRAME           30
#define TRIGGER_ON     1 // 触发模式打开，则 发送一次软触发，采集一次。
#define TRIGGER_OFF     0 // 触发模式关闭，在 每帧开始，自动发送软触发？
#define CONTINUE_ON 1 // 连续采集开启
#define CONTINUE_OFF 0
#define START_GRABBING_ON   1 // 开始取流
#define START_GRABBING_OFF 0 // 关闭取流
#define IMAGE_NAME_LEN   64
#define SAVE_PATH   "F:\\MVS_Data\\temp\\" // 图片保存路径

using namespace std;
using namespace cv;

class LaserVisionSensor : public QWidget
{
```

```
Q_OBJECT

public:
LaserVisionSensor(QWidget* parent = nullptr);
~LaserVisionSensor();

private:
Ui::LaserVisionSensorClass ui;

public:
CMvCamera* m_pcMyCamera[1];         // 相机指针对象 数组    MAX_DEVICE_NUM = 256

MV_CC_DEVICE_INFO_LIST m_stDevList; // 设备信息列表 结构体，用来存储设备列表

cv::Mat* myImage_Camera; // 输出图像缓存 _ 连续采集模式 + 单帧采集模式
cv::Mat* myImage_ScreenShot; // 抓取图像缓存

int devices_num;
string latestSaveName; // 上一次保存的图像名字

/*状态 Status*/
bool  m_bOpenDevice;                        // 是否打开设备 | Whether to open device
bool  m_bStartGrabbing;                     // 是否开始取流 | Whether to start grabbing
int   m_nTriggerMode;                      // 是否是 连续采集模式 | Trigger Mode
int   m_bContinueStarted;                   // 开启过 连续采集图像
MV_SAVE_IAMGE_TYPE   m_nSaveImageType;      // 保存图像格式 | Save Image Type

MyCameraThread* Thread_Camera1 = NULL; // 相机线程对象


private slots:
void OnBnClickedEnumButton();                    // 枚举设备
void OnBnClickedOpenButton();               // 打开设备 | Open device
void OnBnClickedCloseButton();              // 关闭设备 | Close Devices

// Qlable 显示图像
void display_Camera(const Mat* imagePtr, int cameraIndex); // 相机显示
void display_ScreenShot(); // 抓图显示

/* 图像采集 | Image Acquisition*/
void OnBnClickedContinusModeRadio();        // 连续模式 | Continus Mode
void OnBnClickedTriggerModeRadio();         // 触发模式 | Trigger Mode
void OnBnClickedStartGrabbingButton();      // 开始采集 | Start Grabbing
void OnBnClickedStopGrabbingButton();       // 结束采集 | Stop Grabbing

/* 图像保存 | Image Save */
```

```
void OnBnClickedSaveBmpButton();            // 保存 bmp | Save bmp
void OnBnClickedSaveJpgButton();            // 保存 jpg | Save jpg
void OnBnClickedSavePngButton();          // 保存 Png | Save png

/* 焊缝图像处理 */
void OnBnClickedImageProcessButton();
void WeldRecognizeCallback();

private:
void OpenDevices();                    // 打开设备 | Open device
void CloseDevices();                   // 关闭设备 | Close Device
void SaveImage_Cv();                      // 保存图片 | Save Image
void SaveImage_formBuffer();


private slots:
/* 设置、获取参数操作 */
void SetTriggerMode(int m_nTriggerMode);    // 设置触发模式 | Set Trigger Mode
int GetTriggerMode();

void SetExposureTime();                    // 设置曝光时间 | Set Exposure Time
int GetExposureTime();                     // 获取曝光时间 | Get Exposure Time
//void SetGain();                          // 设置增益 | Set Gain
int GetGain();                             // 获取增益 | Get Gain
//void SetFrameRate();                     // 设置帧率 | Set Frame Rate
int GetFrameRate();                        // 获取帧率 | Get Frame Rate

signals:
void singal_Camera_Display(const Mat* image, int index);
void singal_ScreenShot_Display(const Mat* image, int index);
};

#endif // LASERVISIONSENSOR_H
```

MyCameraThread 线程类

继承自 QT 的 QThread，重写 run()，在新线程中循环发送软触发信号，并读取相机内部缓存，将缓存画面显示在界面上

```
void MyCameraThread::run()
{
if (cameraPtr == NULL) return;
if (imagePtr == NULL) return;

while (!isInterruptionRequested())
{
```

```
        // 相机发送软触发信号，并输出返回值
std::cout << "Thread_Trigger:" << cameraPtr->softTrigger() << std::endl;
        // 读取相机中的图像   成功：返回 0   失败：返回-1
std::cout << "Thread_Readbuffer:" << cameraPtr->ReadBuffer(*imagePtr) << std::endl;
/* emit mess();*/
emit singal_Camera_Display(imagePtr, cameraIndex); // 发送信号，主线程接收图像 imagePtr 并显示
msleep(20);
}
}
```

CMvCamera 相机类

以相机的 C++接口为基础，对常用函数进行二次封装，方便用户使用

```
#ifndef _MV_CAMERA_H_
#define _MV_CAMERA_H_

#ifndef MV_NULL
#define MV_NULL   0
#endif

#include "MvCameraControl.h"       // 包含所有的 include 文件
#include <string.h>

#include "opencv2/opencv.hpp"
#include "opencv2/imgproc/types_c.h"

using namespace cv;

class CMvCamera
{
public:
   CMvCamera();
   ~CMvCamera();

   // 获取 SDK 版本号
   static int GetSDKVersion();

   // 枚举设备 | Enumerate Device
   static int EnumDevices(unsigned int nTLayerType, MV_CC_DEVICE_INFO_LIST*
pstDevList);

   // 判断设备是否可达 | Is the device accessible
   static bool IsDeviceAccessible(MV_CC_DEVICE_INFO* pstDevInfo, unsigned int
nAccessMode);
```

```
    // 打开设备 | Open Device
    int Open(MV_CC_DEVICE_INFO* pstDeviceInfo);

    // 关闭设备 | Close Device
    int Close();

    // 判断相机是否处于连接状态 | Is The Device Connected
    bool IsDeviceConnected();

    // 注册图像数据回调 | Register Image Data CallBack
    int RegisterImageCallBack(void(__stdcall* cbOutput)(unsigned char* pData,
MV_FRAME_OUT_INFO_EX* pFrameInfo, void* pUser), void* pUser);

    // 开启抓图 | Start Grabbing
    int StartGrabbing();

    // 停止抓图 | Stop Grabbing
    int StopGrabbing();

    // 主动获取一帧图像数据 | Get one frame initiatively
    int GetImageBuffer(MV_FRAME_OUT* pFrame, int nMsec);

    // 释放图像缓存 | Free image buffer
    int FreeImageBuffer(MV_FRAME_OUT* pFrame);

    // 主动获取一帧图像数据 | Get one frame initiatively
    int GetOneFrameTimeout(unsigned char* pData, unsigned int* pnDataLen, unsigned int
nDataSize, MV_FRAME_OUT_INFO_EX* pFrameInfo, int nMsec);

    // 显示一帧图像 | Display one frame image
    int DisplayOneFrame(MV_DISPLAY_FRAME_INFO* pDisplayInfo);

    // 设置 SDK 内部图像缓存节点个数 | Set the number of the internal image cache nodes in
SDK
    int SetImageNodeNum(unsigned int nNum);

    // 获取设备信息 | Get device information
    int GetDeviceInfo(MV_CC_DEVICE_INFO* pstDevInfo);

    // 获取 GEV 相机的统计信息 | Get detect info of GEV camera
    int GetGevAllMatchInfo(MV_MATCH_INFO_NET_DETECT* pMatchInfoNetDetect);

    // 获取 U3V 相机的统计信息 | Get detect info of U3V camera
    int GetU3VAllMatchInfo(MV_MATCH_INFO_USB_DETECT* pMatchInfoUSBDetect);
```

```
    // 获取和设置 Int 型参数，如 Width 和 Height，详细内容参考 SDK 安装目录下的
MvCameraNode.xlsx 文件
    // int GetIntValue(IN const char* strKey, OUT MVCC_INTVALUE_EX* pIntValue);
    int GetIntValue(IN const char* strKey, OUT unsigned int* pnValue);
    int SetIntValue(IN const char* strKey, IN int64_t nValue);

    // 获取和设置 Enum 型参数，如 PixelFormat，详细内容参考 SDK 安装目录下的
MvCameraNode.xlsx 文件
    int GetEnumValue(IN const char* strKey, OUT MVCC_ENUMVALUE* pEnumValue);
    int SetEnumValue(IN const char* strKey, IN unsigned int nValue);
    int SetEnumValueByString(IN const char* strKey, IN const char* sValue);

    // 获取和设置 Float 型参数，如 ExposureTime 和 Gain，详细内容参考 SDK 安装目录下的
MvCameraNode.xlsx 文件
    int GetFloatValue(IN const char* strKey, OUT MVCC_FLOATVALUE* pFloatValue);
    int SetFloatValue(IN const char* strKey, IN float fValue);

    // 获取和设置 Bool 型参数，如 ReverseX，详细内容参考 SDK 安装目录下的 MvCameraNode.xlsx
文件
    int GetBoolValue(IN const char* strKey, OUT bool* pbValue);
    int SetBoolValue(IN const char* strKey, IN bool bValue);

    // 获取和设置 String 型参数，如 DeviceUserID，详细内容参考 SDK 安装目录下的
MvCameraNode.xlsx 文件 UserSetSave
    int GetStringValue(IN const char* strKey, MVCC_STRINGVALUE* pStringValue);
    int SetStringValue(IN const char* strKey, IN const char* strValue);

    // 执行一次 Command 型命令，如 UserSetSave，详细内容参考 SDK 安装目录下的
MvCameraNode.xlsx 文件
    int CommandExecute(IN const char* strKey);

    // 探测网络最佳包大小(只对 GigE 相机有效)
    int GetOptimalPacketSize(unsigned int* pOptimalPacketSize);

    // 注册消息异常回调
    int RegisterExceptionCallBack(void(__stdcall* cbException)(unsigned int nMsgType,
void* pUser), void* pUser);

    // 注册单个事件回调
    int RegisterEventCallBack(const char* pEventName, void(__stdcall*
cbEvent)(MV_EVENT_OUT_INFO* pEventInfo, void* pUser), void* pUser);

    // 强制 IP | Force IP
    int ForceIp(unsigned int nIP, unsigned int nSubNetMask, unsigned int
nDefaultGateWay);
```

```
    // 配置 IP 方式 | IP configuration method
    int SetIpConfig(unsigned int nType);

    // 设置网络传输模式 | Set Net Transfer Mode
    int SetNetTransMode(unsigned int nType);

    // 像素格式转换 | Pixel format conversion
    int ConvertPixelType(MV_CC_PIXEL_CONVERT_PARAM* pstCvtParam);

    // 保存图片 | save image
    int SaveImage(MV_SAVE_IMAGE_PARAM_EX* pstParam);

    // 保存图片为文件 | Save the image as a file
    int SaveImageToFile(MV_SAVE_IMG_TO_FILE_PARAM* pstParam);

    // 设置是否为触发模式
    int setTriggerMode(unsigned int TriggerModeNum);

    // 设置触发源
    int setTriggerSource(unsigned int TriggerSourceNum);

    // 软触发
    int softTrigger();

    // 读取 buffer
    int ReadBuffer(cv::Mat& image);

public:
    void* m_hDevHandle;                         // 设备句柄
    unsigned int    m_nTLayerType;              // LayerType

public:
    // ReadBuffer() 接收缓存
    unsigned char*  m_pBufForSaveImage;         // 输出图像缓存
    unsigned int    m_nBufSizeForSaveImage;

    unsigned char*  m_pBufForDriver;
    unsigned int    m_nBufSizeForDriver;
};

#endif      //_MV_CAMERA_H_
```

WeldCom 图像处理算法类

自己提出的 针对交叉结构光的识别算法类

```
#ifndef T_COMMON_H
#define T_COMMON_H

#include <opencv2\opencv.hpp>
#include <iostream>

using namespace std;
using namespace cv;

class WeldCom
{
public:
WeldCom(Mat& src);
~WeldCom();

Point CrossPointlocation_step1(Mat&); /* 交叉点 初略定位*/
void CrossPointlocation_step2(Mat&, int); /* 对搜索小窗口内进行 交叉点定位*/
void edgePointlocation(Mat& src); /* 边缘点定位*/
static bool findLight(Mat& binary, vector<int>& lightIdx, string mode); /* 检测光条存在的行列索引*/

void weldLine(Mat& src);          /* 显示识别结果*/


private:
void regionSegment(Mat&);         /* 四区域分割*/
Mat Multidiagonal(int lw);        /* 创建卷积核矩阵*/
vector<Point> wds_Roi(Mat& roi, int upperRowIndex); /* 垂直灰度重心法 , 在 CrossPointlocation_step1 中 拟合前*/
bool gapNum(Mat& roi, int T);     /* 检测是否存在间隙（阈值 = 2 pixel）*/

void typeArray();                 /* 焊缝类型数组 */
float weldWidth(Point, Point);    /* 焊缝宽度*/


public:
Point crossPoint;                 /* 交叉点坐标*/
vector<Mat> regionROI;            /* 存储四个 ROI 子图*/
Point startP1;                    /* laser1 V 型焊缝 点集分割 要用到的起点 (下边)*/
Point startP2;                    /* laser2 V 型焊缝 点集分割 要用到的起点 (上边)*/
Mat laser1;                             /* laser1 单线 (下边)*/
Mat laser2;                             /* laser2 单线 (上边)*/

Point edgeP1;                           /* laser1 工件边缘点 (下边)*/
Point edgeP2;                           /* laser2 工件边缘点 (上边)*/
Mat laser1_OnWorkpiece;                 /* 边缘点左边的工件光条图像*/
```

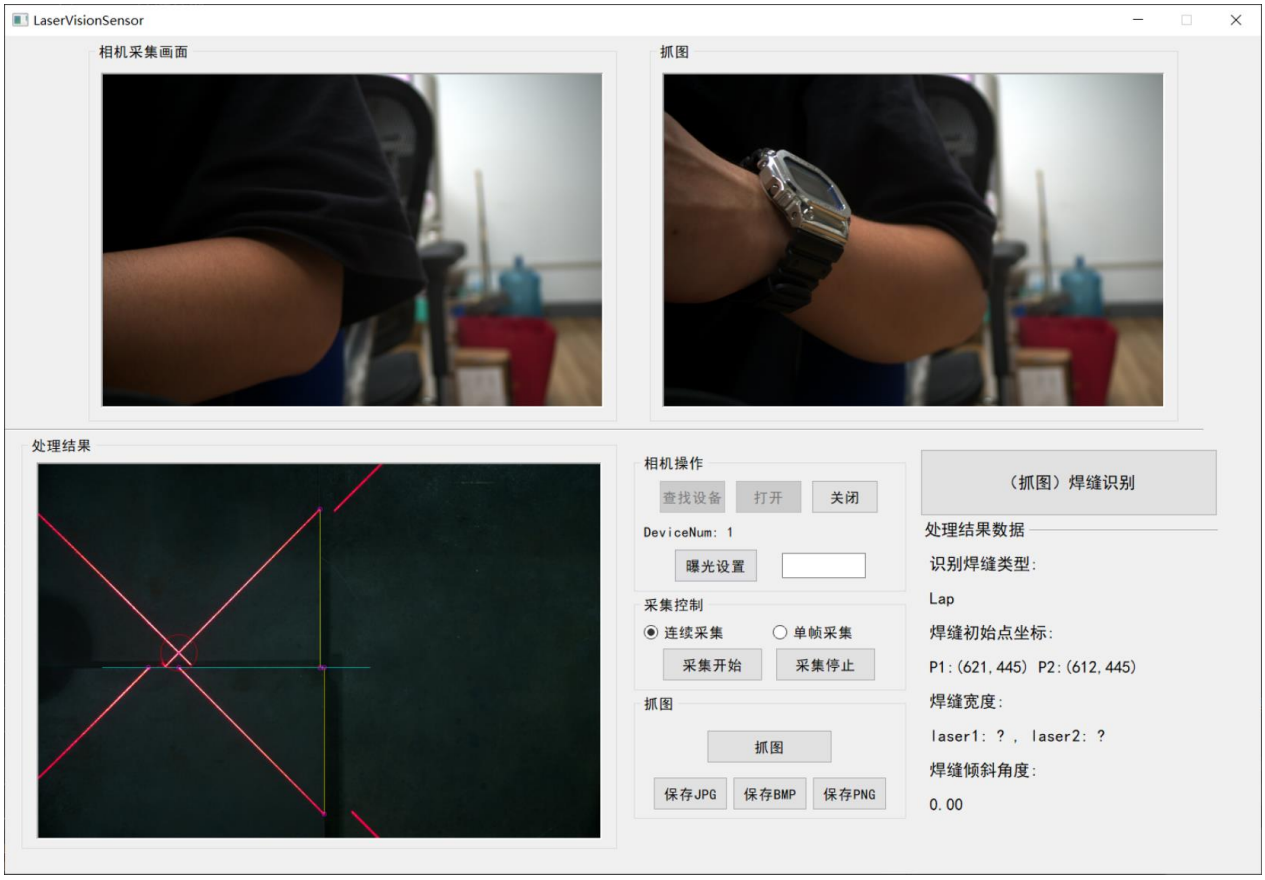
```
Mat laser2_OnWorkpiece;

string wledType;                        /* 识别的焊缝类型*/
Point initP1;                           /* 初始点坐标*/
Point initP2;
double weldWidth1;                      /* 焊缝坡口宽度*/
double weldWidth2;
double degree;                          /* 焊缝倾斜角度*/
Point weldP1;                           /* laser1 焊缝特征点 */
Point weldP2;                           /* laser2 焊缝特征点*/
};

#endif // T_COMMON_H
```

QT 界面

界面包括：

1. 相机采集画面（上左）：在连续采集模式下，点击采集开始按钮，相机开始取流，并将读取的图像实时显示到该画面中；在单帧采集模式下，点击采集开始按钮，相机开始取流，将一帧画面显示在画面中，随后自动关闭取流。
2. 抓图画面（上右）：显示感兴趣的图像进行抓取并显示在该区域，该画面图像会保存在缓存中，便于保存为本地图像文件，以及后续的图像处理。
3. 相机操作区（下中）：包含基本的相机操作按钮 — 查找相机设备、打开关闭、曝光设置、相机采集模式控制、开始和停止取流。并提供了抓图功能，并可以将抓取的图像保存为不同图像格式的文件。
4. 处理结果数据（下右）：点击开始焊缝识别按钮，就会调用自己写的交叉结构光的处理算法，来识别焊缝的相关参数。并将处理结果显示在界面上。
5. 处理结果图像（下右）：将焊缝图像的处理结果显示。

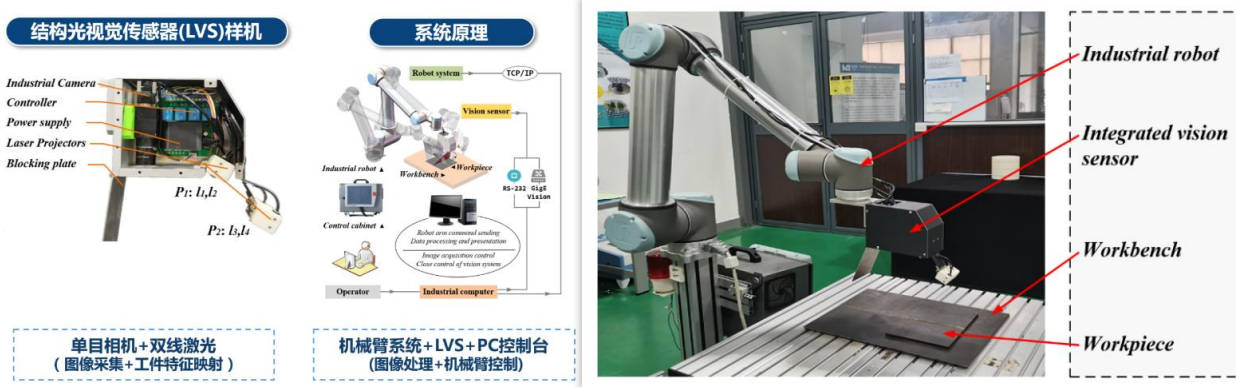
注：上两图为相机测试画面，下面的焊缝识别图像为本地测试图像识别结果。



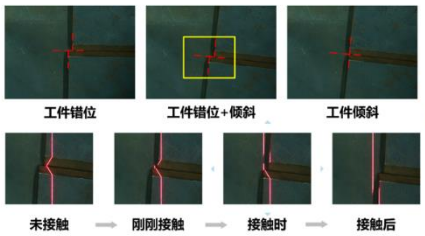
图像处理流程

图像处理过程就是自己研究领域的一些实现算法，下面是一些简要的提出的算法的相关流程。

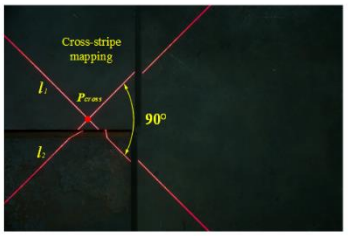
相关处理流程，通过 C++代码形式，在 QT 中进行调用处理。



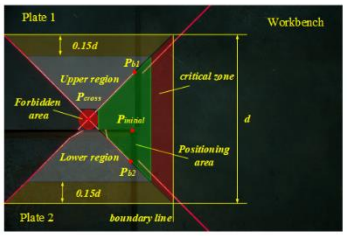
一种基于交叉结构光的多类型焊缝初始点定位方法



- 适用多种焊缝类型
- 包容工件错位、倾斜
- 无需移动扫描，直接识别定位
- 定位精度高（<0.5mm）
- 判断对齐、报错
- 焊缝参数获取(焊缝方向、宽度)



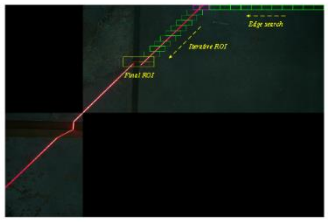
交叉结构光 映射图案



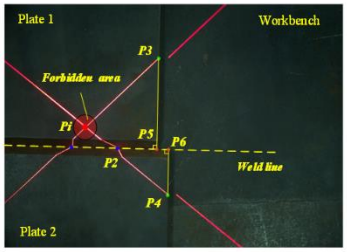
工件摆放区域

初始点识别过程

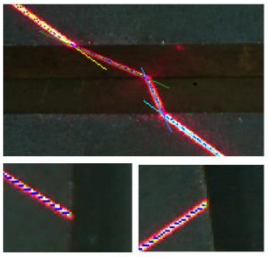
交叉线分割



工件边缘点提取

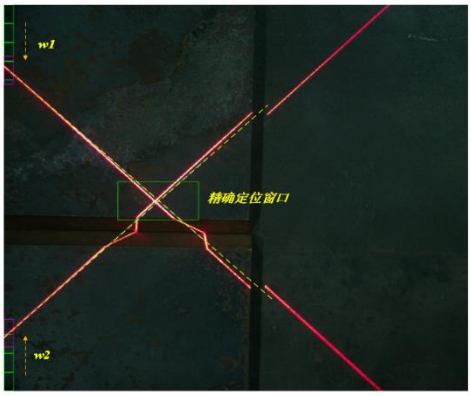


基于几何的初始点识别



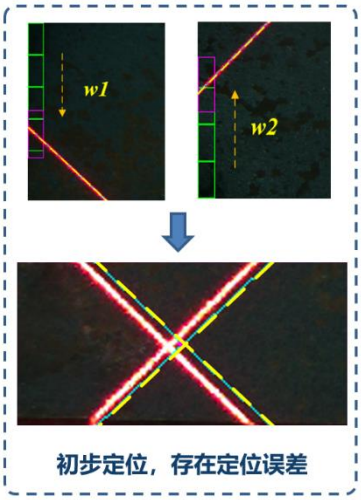
特征点提取
(焊缝特征点+工件边界点)

Step1：基于交叉线点定位的交叉光条分割



✓ 搜索定位：局部处理，减少计算量，排除噪声干扰

✗ 全局定位：全图处理，引入噪声干扰



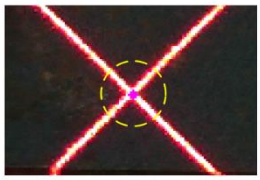
初步定位，存在定位误差

Step1：基于交叉线点定位的交叉光条分割

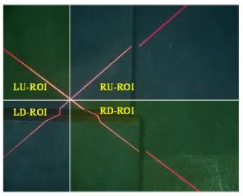
$$T(i,j)=\sum_{k=-1}^{1}\left[I(i,j+k)+I(i+k,j\pm2)-I(i+k,j\pm5)\right]+\sum_{m=-2}^{2}\sum_{n=0}^{2}I(i+m,j\pm(n+3))-I(i,j\pm4)$$

式中$(i=2,3,\cdots,row-3;\ j=5,6,\cdots,col-6)$

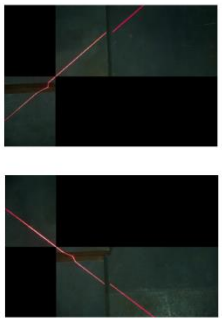
交叉点定位算子



交叉点定位效果
（定位精度在2个像素以内）

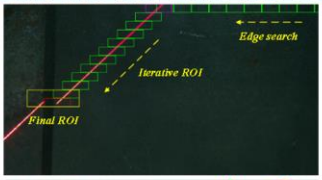


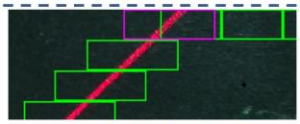
4个ROI子区域



分割效果

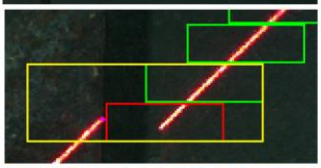
Step2：工件边缘点的迭代搜索



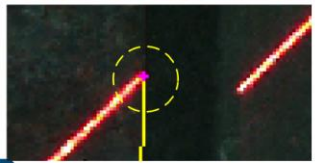


初始迭代ROI（紫框）

光条间断判断：
1. 条纹宽度 > 设定值 T1
2. 条纹宽度 < 设定值 T2
3. 条纹高度 < 设定值 h
4. 当前窗口 未检测到光条

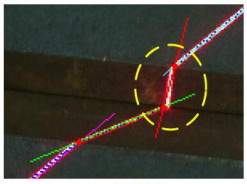


最终迭代ROI（红框）| 边缘点定位ROI（黄框）

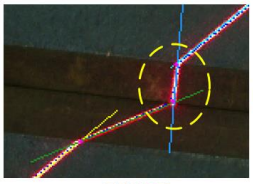


交叉点定位效果
（定位精度在1个像素）

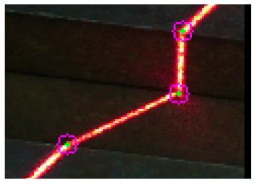
初始点定位精度（V形）



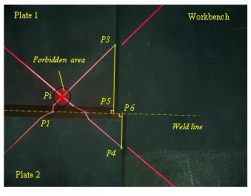
改进前
（单特征点误差在2个像素以上）



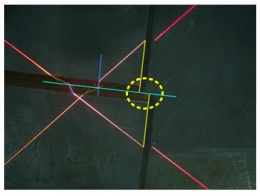
改进后
（单特征点精度在1个像素之内）



焊缝特征点



（提取示意方法图）

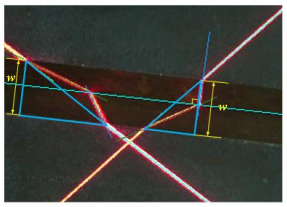


交叉结构光提取结果

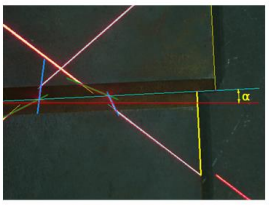
最终初始点定位效果
（XY方向精度均在2个像素左右）
（ <=0.25mm）

焊缝参数获取（V形）

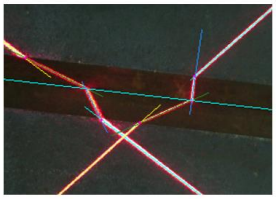
1.初始点的精确位置



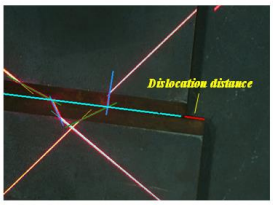
2.焊缝坡口宽度



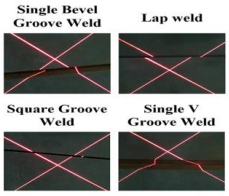
3. 焊缝角度



4.焊缝特征点的深度
(结合标定的视觉模型)



5.工件错位距离
(提示未对齐)



6.焊缝分类