華中科技大學

数字图像处理和模式识别

综合实验报告

工件识别与分类

学　　号：	U201917293、U201914749
姓　　名：	房江祎、韦思成
指导老师：	郑定富、马杰
院　　系：	人工智能与自动化学院
专　　业：	自实 1901

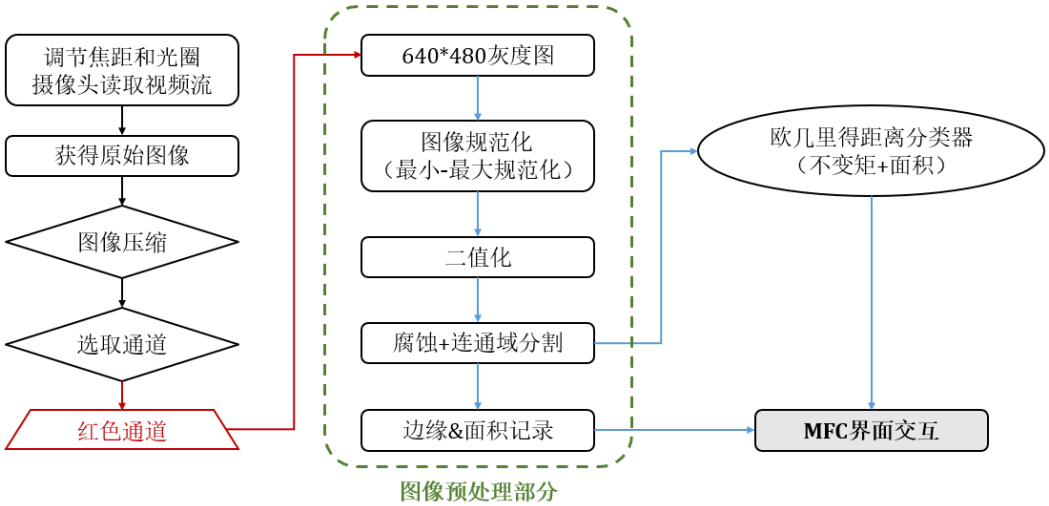
自动化学院

2022 年 4 月 9 日

目录

一、总体方案设计

总体方案的主要流程如下图所示：



图片 1　总体方案流程图

我们的整体方案主要包括四个环节，即：摄像设备的硬件交互环节、图像预处理环节、分类器环节、MFC 界面交互环节。

第一， **摄像设备的硬件交互环节**，由房江祎实现。这部分的主要目的是获取较为理想的帧图像（即 640×480 的灰度图），并将其传至下一环节。根据相机开发手册，在硬件层面，我们利用 MVGige 头文件中的 API，帮助我们从控制相机，以及在线调整相机帧率以及增益等；在软件层面，我们利用 MVImage 头文件中的 API，帮助我们创建符合相机像素格式的图片包括完成图片比例缩放等。

第二， **图像预处理环节**，由韦思成实现。这部分的主要目的是对 640×480 的初始灰度图进行处理，实现对图像进行腐蚀、连通域信息收集、边缘信息收集，也是全方案效率的重要保证。在本环节中，我们灵活运用了大量静态数组、堆栈并实现了高耦合度的算法，将本环节的时间复杂度和空间复杂度均控制在$O(n)$。目前，本环节算法的效率在面对 640×480 的图像可以在 30ms 内完成。同时由于我们对灰度图进行了标准化和规范化处理，其鲁棒性也得到了保证。

第三， **分类器环节**，这部分由房江祎和韦思成共同实现。这部分主要使用了图像预处理环节得到的数据，即各连通域（工件）的不变矩和面积。在本环节中，我们将不变矩和面积映射入二维平面中，并与各标准件的数据进行对比，搜索出距离最相近的标准点以实现分类。

第四， **MFC 界面交互环节**，由房江祎实现。在设计过程中，我们遵循依次进行状态设计，跳转条件设计，状态执行操作设计的步骤；我们遵循精简，人性化，美观的，直观等的原则。

总体来说，我们实现了一个高实时性（＞15 FPS）、界面可交互性较好的工件分类软件。

二、关键技术

2.1 压缩单通道灰度图的提取

本项目使用的摄像设备关键参数如下：

型号	最高分辨率	最大帧率	输出颜色	数据位数	可编程控制
MV-EM120M/C	1280 × 960	40fps	黑白/彩色	8/12	增益、帧率、曝光时间

根据相机开发手册，在硬件层面，我们利用 MVGige 头文件中的 API，帮助我们从控制相机，以及在线调整相机帧率以及增益等；在软件层面，我们利用 MVImage 头文件中的 API，帮助我们创建符合相机像素格式的图片包括完成图片比例缩放等。

其中对于摄像头高度，焦距，镜头光圈的调整则是以清晰，人能进行准确的识别为准。

出于实时性与准确性折衷的角度，我选择将从相机1280 × 960的分辨率格式直接读取的图像转换为640 × 480的分辨率格式图像。为了同时支持摄像机访问与文件操作访问（统一API），我选择重载了两种图片预处理函数，分别与两种访问形式对应。（对于支持文件操作访问的自定义图片缩放函数详情见代码）

```
void ImgSys::preprocess(HANDLE cam, MV_IMAGE_INFO* pInfo, int zoomrate)
{
    int w, h;
    MV_PixelFormatEnums m_PixelFormat;
    MVImage tmpimage;
    MVGetWidth(cam, &w);
    MVGetHeight(cam, &h);
    MVGetPixelFormat(cam, &m_PixelFormat);
    tmpimage.CreateByPixelFormat(w, h, m_PixelFormat);//创建临时图片存储区
    if (!imgdatabase->Isempty(0))
    {
        imgdatabase->deleteimage(0);
        imgdatabase->newimage(0);
    }
    if(imgArr==NULL)
        imgArr = imgdatabase->newarray(h/zoomrate, w/zoomrate);//创建缩放后内存空间
    imgdatabase->newimage(h/zoomrate,w/zoomrate,m_PixelFormat,0);
    imgdatabase->newimage(h/zoomrate, w/zoomrate, PixelFormat_Mono8, 1);
    MVInfo2Image(cam, pInfo, &tmpimage);
    MVGetPixelFormat(cam, &m_PixelFormat);
    MVZoomImageBGR(cam, (uchar*)tmpimage.GetBits(),
        w, h, (uchar*)imgdatabase->getImageData(0),
        1 / (double)zoomrate, 1 / (double)zoomrate);//利用自带缩放函数
    imgdatabase->threec2onec();
    tmpimage.Destroy();
}
```

```
void ImgSys::preprocess(CString strFilePath, int zoomrate)
{
    MV_PixelFormatEnums pm;    MVImage tmpimage;
    if (!imgdatabase->Isempty(0)){
        imgdatabase->deleteimage(0);
        imgdatabase->newimage(0);
    }
    if (!imgdatabase->Isempty(1)){
        imgdatabase->deleteimage(1);
        imgdatabase->newimage(1);
    }
    tmpimage.Load(strFilePath);
    const int h = tmpimage.GetHeight();const int w = tmpimage.GetWidth();
    const int channels = tmpimage.GetBPP()/8;
    if (channels == 3)//进行预处理前图片像素格式判断
        pm = PixelFormat_BayerBG8;
    else
        pm = PixelFormat_Mono8;
    if (imgArr == NULL)
        imgArr = imgdatabase->newarray(h / zoomrate, w / zoomrate);
    imgdatabase->newimage(h / zoomrate, w / zoomrate, PixelFormat_Mono8, 1);
    imgdatabase->newimage(h/ zoomrate, w/ zoomrate, pm, 0);
    uchar* srcp = (uchar*)tmpimage.GetBits();uchar* dstp = (uchar*)imgdatabase->getImageData(0);
    ZoomInImg(srcp, dstp, w, h, channels, zoomrate);//自行编写的缩放函数
    imgdatabase->threec2onec();tmpimage.Destroy();
}
```

图片 2　图像分辨率转换

利用相机产商提供 API 获得指向图片内存数据指针，其内存组织方式为将彩色三通道宽高二维图像，沿 BGR 先宽后高展开，核心转换公式：imgArr[i][j] = *(p + j * bpp / 8 + REDCHAN + i * oneline);其中，j 为宽度方向指针，i 为高度方向指针，oneline 为一行对应字节数，REDCHAN 代表红色通道偏移量。

2.2 高实时性与高耦合度的图像预处理环节

如图 1 流程图所示，图像预处理部分是辅助决策的关键部分。

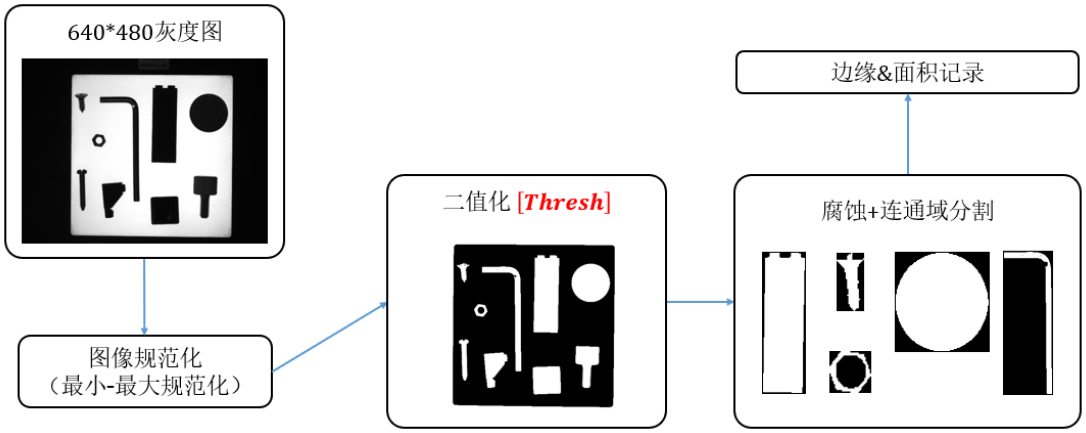
一个优秀的图像处理环节应当能够快速提取出单个工件大量特征信息，例如边缘、面积等。同时，除了时间复杂度之外，还需要将空间复杂度限制在可控范围内。

在这个过程中，我们必须保证图像预处理的高实时性。由于图像包含的像素点极多，哪

怕是单次遍历都会造成一定的时间损耗，因此必须保证全过程的时间复杂度控制在$O(n)$以内（n表示像素数目），也必须保证遍历单次的情况下获取尽可能多的数据信息。

因此，预处理部分的核心思想就是“空间换时间”。**这部分的实现主要在 ImgAlgo 类中**。

图像预处理环节的主要流程如下：



图片 3　图片预处理环节的主要流程图

在本环节中，有以下的技巧来加速算法：

1、尽可能减少动态数组的生成

```
static int regA[BD_SIZE][2];     // 边缘寄存器A
static int regB[BD_SIZE][2];     // 边缘寄存器B
static int area[SUP][2];         // 区域寄存器
static CPoint border[BD_SIZE];   // 边界寄存器
```

例如上述代码，为了避免在搜索连通域过程中间断地动态分配数组空间造成的时间损耗，我们直接在 ImgAlgo 类中声明了大量较长的静态寄存器数组，方便进行遍历边缘、区域、工件边界的存储，较为明显地提升了速度。

2、广度优先搜索的循环化表示

广度优先搜索较为简洁的写法依然是递归，但是由于递归在深度较大的情况下极容易导致堆栈过大内存溢出，同时其运行速度也会因为堆栈的不断保存收到影响，因此必

```
void fillConnected(uchar** img, int py, int px, int* stats) {
    int index = 0, cnt = 0;
    int ey, ex;
    int(*border)[2] = regA, (*regter)[2] = regB;
    areaClear(area);
    borderClear(border), borderClear(regter);
    border[0][0] = py, border[0][1] = px;
    while (border[0][0] > -1) {
        while (border[index][0] > -1) {
            ey = border[index][0], ex = border[index][1];
            isIn = false, atEdge(ey, ex);
            if (--ex > -INFTY && edge[0] && isBorder(img[ey][ex], ey, ex) && img[ey][ex] > CHG_SIGN) regter[cnt][0] = ey, regter[cnt++][1] = ex, updStats(img, ey, ex, stats);
            if (++ey > -INFTY && edge[1] && isBorder(img[ey][ex], ey, ex) && img[ey][ex] > CHG_SIGN) regter[cnt][0] = ey, regter[cnt++][1] = ex, updStats(img, ey, ex, stats);
            if (++ex > -INFTY && edge[2] && isBorder(img[ey][ex], ey, ex) && img[ey][ex] > CHG_SIGN) regter[cnt][0] = ey, regter[cnt++][1] = ex, updStats(img, ey, ex, stats);
            if (++ex > -INFTY && edge[3] && isBorder(img[ey][ex], ey, ex) && img[ey][ex] > CHG_SIGN) regter[cnt][0] = ey, regter[cnt++][1] = ex, updStats(img, ey, ex, stats);
            if (--ey > -INFTY && edge[4] && isBorder(img[ey][ex], ey, ex) && img[ey][ex] > CHG_SIGN) regter[cnt][0] = ey, regter[cnt++][1] = ex, updStats(img, ey, ex, stats);
            if (--ey > -INFTY && edge[5] && isBorder(img[ey][ex], ey, ex) && img[ey][ex] > CHG_SIGN) regter[cnt][0] = ey, regter[cnt++][1] = ex, updStats(img, ey, ex, stats);
            if (--ex > -INFTY && edge[6] && isBorder(img[ey][ex], ey, ex) && img[ey][ex] > CHG_SIGN) regter[cnt][0] = ey, regter[cnt++][1] = ex, updStats(img, ey, ex, stats);
            if (--ex > -INFTY && edge[7] && isBorder(img[ey][ex], ey, ex) && img[ey][ex] > CHG_SIGN) regter[cnt][0] = ey, regter[cnt++][1] = ex, updStats(img, ey, ex, stats);
            index++;
        }
        borderClear(border);
        border = (border == regA) ? regB : regA;
        regter = (regter == regA) ? regB : regA;
        index = 0, cnt = 0;
    }
}
```

图片 4　广度优先搜索的循环化核心代码

须使用循环的方式实现广度优先搜索。这里我们对二值图像利用了条件判断的性质并引入了一个中间变量 CHG_SIGN 以加快循环速度。

3、“并发”式的工件数据处理方式

由图 3 中，在单次遍历属于工件的像素点时，我们均运行了 updStats 函数。在这个函数中，我们“并发”地处理了大量数据。包括收集工件方框的边界、记录工件的每一个坐标及面积。这样保证了在减少代码量和运行时间的同时，收集各工件的边界数据。

```
void updStats(uchar** img, int ey, int ex, int* stats) {
    img[ey][ex] = CHG_SIGN;
    // 获取工件方框的数据
    if (ey < stats[0]) stats[0] = ey;
    if (ey > stats[1]) stats[1] = ey;
    if (ex < stats[2]) stats[2] = ex;
    if (ex > stats[3]) stats[3] = ex;          // 矩形边界数据
    stats[4]++;                                // 面积数据
    area[pt][0] = ey, area[pt][1] = ex;        // 记录工件的每一个坐标
    pt++;
    if (pt == SUP) pt--;
}
```

同时，在遍历地搜索每个工件点时，我们也同时运行了isBorder函数来专门收集工件的边缘特性，函数如下：

```
bool isBorder(uchar val, int py, int px) {
    if (bt == BD_SIZE - 1) bt--;
    if (!isIn && !val) isIn = true, border[bt].y = py, border[bt++].x = px;
    return true;
}
```

2.3 不变矩综合分类器

HU 矩是由 Hu (*Visual pattern recognition by monent invariants*) 在 1962 年提出的：

$$\text{原点矩：}\ m_{pq} = \sum_{x=1}^{M}\sum_{y=1}^{N} x^p y^q f(x,y) \tag{1}$$

$$\text{中心矩：}\ \mu_{pq} = \sum_{x=1}^{M}\sum_{y=1}^{N} (x-x_0)^p (y-y_0)^q f(x,y) \tag{2}$$

$$\text{归一化中心矩：}\ y_{pq} = \frac{\mu_{pq}}{\mu_{00}^{r}}\text{，其中，}\ r = \frac{p+q+2}{2}, p+q = 2,3,4\ldots$$

当图像发生变化时，m_{pq}也发生变化，而μ_{pq}则具有平移不变性，但对旋转依然敏感。如果用归一化中心矩，则特征不仅具有平移不变性，而且具有比例不变性。

这里我们使用的就是特殊的归一化中心矩：

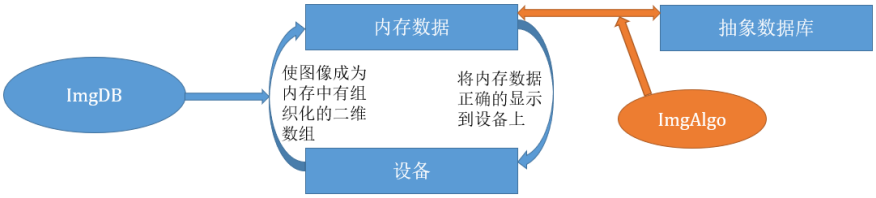
$$T = y_{02} + y_{20} \tag{3}$$

确定了计算方法后，我们将各个工件（也就是分出的各个连通域）的不变矩计算耦合到了切割图片的算法中，更好地节省了时间。

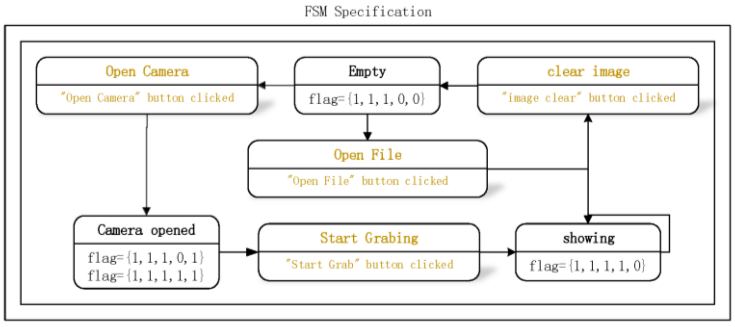
```
while (area[index][0] > -1) {
    py = area[index][0] - object.ymin;
    px = area[index][1] - object.xmin;
    workpiece[py][px] = 255;
    Mc += 1, Mx += px, My += py;
    index++;
}
Mx /= Mc, My /= Mc;
index = 0;
while (area[index][0] > -1) {
    py = area[index][0] - object.ymin;
    px = area[index][1] - object.xmin;
    rule += pow(py - My, 2) + pow(px - Mx, 2);
    index++;
}
rule = rule / Mc / Mc;
rule_inv = rule;
```

2.4 MFC 交互部分的亮点

图片数据库层为内存数据与设备间提供可靠的数据传输与维护。**图片数据库层**为**抽象算法层**提供内存数据组织化服务。**抽象算法层**调用**图片数据库**提供的服务完成数据组织化，同时抽象算法服务将从数据中提取抽象信息并转化为底层内存数据。图片数据库层调用抽象算法层算法服务，同时完成内存数据在屏幕上的显示。



FSM 设计总共有三种状态：空闲（图片展示区空闲等待操作），相机已打开（打开相机等待采集操作），图片展示。状态跳转流程如上图所示，每个状态跳转所采取的操作如上图所示。



同时，为了达到非阻塞响应 UI 的效果——即在 Onstream 回调函数执行的同时，效应鼠标按键同时多次按键覆盖并只响应最近一次操作的行为。我选择维护一个定时器队列，从而实现覆盖、定时结束自清。

```
void CtoolrecognDlg::OnTimer(UINT_PTR nIDEvent)
{
    // TODO: 在此添加消息处理程序代码和/或调用默认值
    state = C;
    if(q.back()==nIDEvent)   //队列中最后一个定时器到时
        m_Point = { -1,-1 };//焦点复位
    KillTimer(nIDEvent);     //删除定时器
    q.pop();                 //弹出队列
    CDialog::OnTimer(nIDEvent);
}
```

```
if (((point.x >= IRect.left) && (point.x <= IRect.right))
    && (point.y >= IRect.top && point.y <= IRect.bottom))//判断鼠标点击坐标是否在图像区
{
    m_Point.x = point.x - IRect.left;
    m_Point.y = point.y - IRect.top;
}
m_Point.x = m_Point.x * imgsys.imgdatabase->getImageWidth(0)  / (IRect.Width());//将鼠标
m_Point.y = m_Point.y * imgsys.imgdatabase->getImageHeight(0) / (IRect.Height());
if (count >= 10000)
    count = 0;
q.push(count);
SetTimer(count, 2000, NULL);
count++;
CDialog::OnLButtonDown(nFlags, point);
}
```

三、图像采集系统设计

3.1 任务概述

- 目的概述

1、实现不同形状工件的检测识别
2、实现工件的自动分类、计数，面积测量

- 任务、传感器分析

1、实现不同形状工件的检测识别
2、实现工件的自动分类、计数，面积测量

- 预期目标/指标

1、不粘连工件高精度识别，粘连工件一定的识别能力
2、较为优秀的实时性和交互界面

3.2 难点分析

1、读取内存和界面响应设计不在于难度而在于繁复。由于从相机获得的图像类是 MVImage 类型，内存数据是顺序存储，其中按照 BGR，先逐列后逐行的顺序进行存放。因此为了便于处理，我们首先自行书写内存组织函数，将连续存放的 BGR 内存数据组织成二维数组形式的单通道灰度图。

2、针对界面响应采取 FSM 有限状态机的设计方式，首先分析出界面有几种状态（等待输入方式接入，等待开始处理指令，等待停止处理指令，等待输入流关闭清空），其次需要分析清楚如何进行界面状态与状态之间的转换。

3、必须保证图像预处理（即分类器前的预处理）的高实时性。由于图像包含的像素点极多，哪怕是单次遍历都会造成一定的时间损耗，因此必须保证全过程的时间复杂度控制在$O(n)$以内（n表示像素数目）。

4、分类器的设计以及其面对粘连问题时的鲁棒性。由于我们决定采用连通域分割的方式对图像进行预处理，面对粘连问题时，分类器必须仍然具备解决问题的能力。这也对我们提取图像数据的算法做出了一定的要求，必须提取出更多有决定性的特征。

3.3 硬件方案

1、高度调整：保证红色背光灯整体在摄像头的视野内（分析面积需要）。

2、焦距调整：保证清晰即可。

3、光源、光圈调整：尽量使背光灯外的背景为肉眼可见的全黑色。

3.4 软件方案

1、摄像设备的硬件交互环节：利用厂商提供的 API 进行图像的读写和缩放。

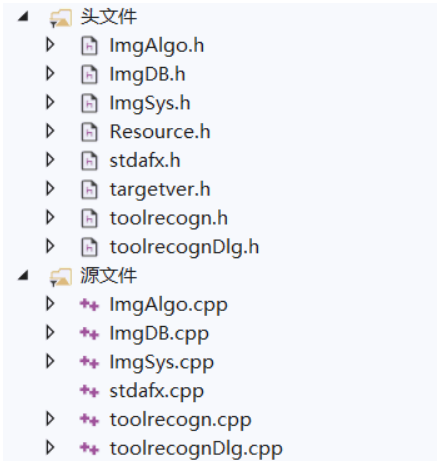
2、图像预处理环节：提取灰度图中的连通域及其关键信息。

3、分类器环节：根据连通域的不变矩和标准化后的面积进行分类。

4、MFC 界面交互环节：采用 FSM 设计非阻塞 UI。

四、源程序设计和运行结果

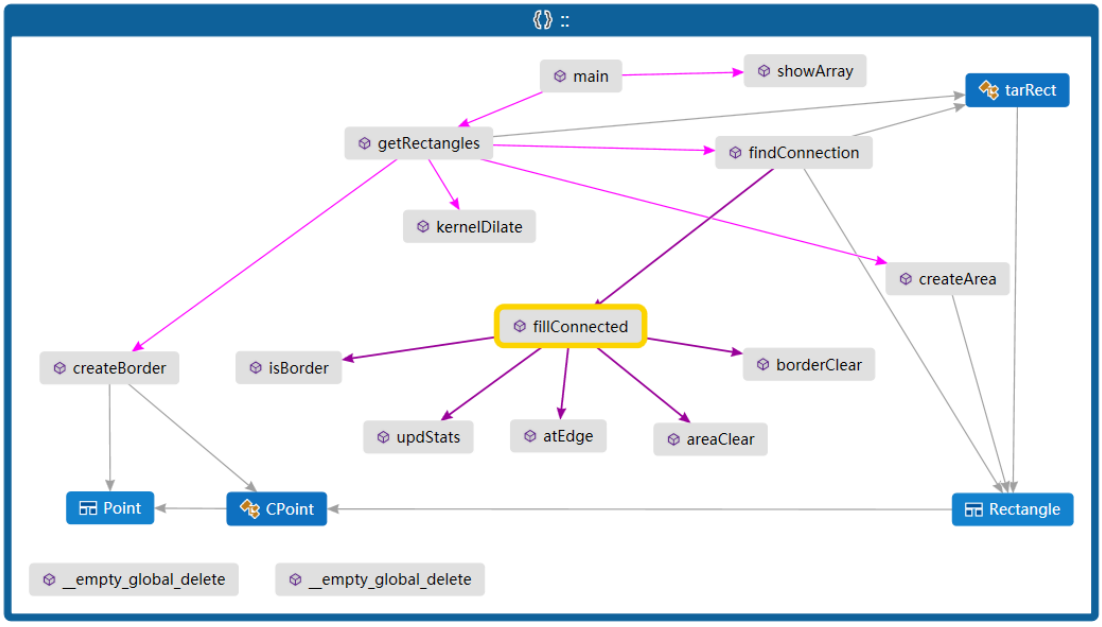
目录部分：



ImgDB 表示图像数据库，提供图像增删改查相关服务给 ImgSys 进行调用。ImgSys 控制 ImgDB 和 ImgAlgo 的数据交互，以及从外界获取响应信息，来确定服务运行逻辑与时机。

ImgAlgo 就是抽象算法库，包含了图像预处理以及工件分类的算法。

ImgAlgo 下的图像预处理函数结构设计：

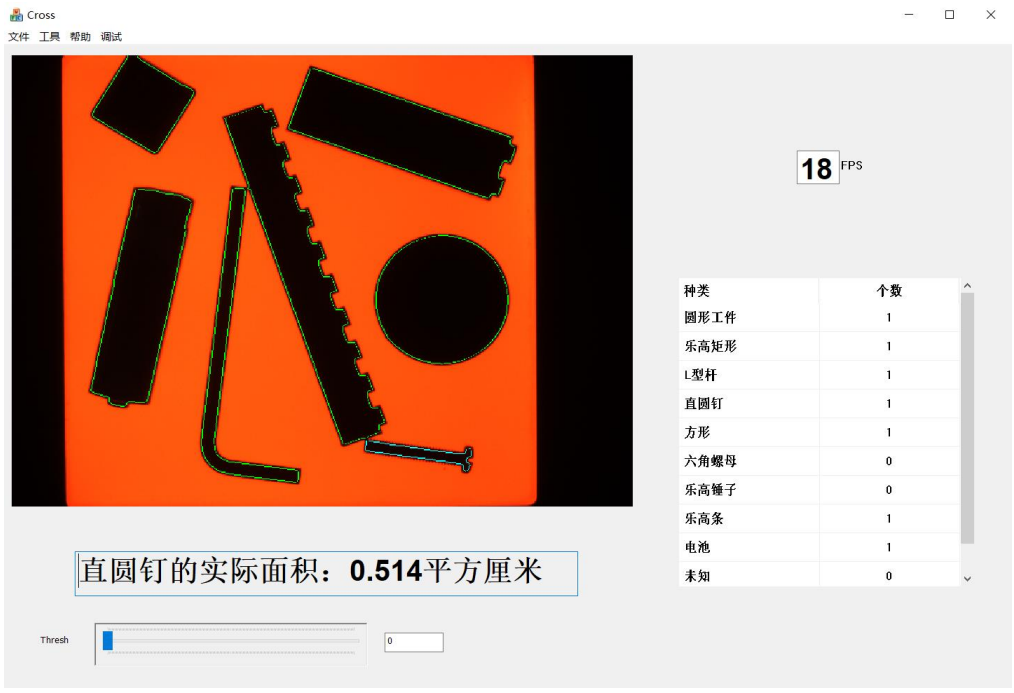


ImgAlgo 的核心函数就是 fillConnected，对单个连通域进行类 BFS 的泛洪填充，同时在不增加时间复杂度的情况下获取该连通域的各个信息。fillConnected 下的所有函数都是辅助函数，其上的所有函数都是集成函数，以实现对全图所有连通域进行数据收集的功能。

本软件 UI 设计创新处有三：一、实时 FPS 显示，放映出软件的高效实时性；二、通过滑块交互式调整二值化阈值，以适应不同外界条件下分类情况，使程序更加鲁棒；三、点击工件将以不同于其余未被点击工件颜色的边缘被绘制，同时在文本框内显示工件的实际面积，更加直观人性化。

五、实验结果及其评价

如下，是两张运行时的截图：



可以看出，程序的实时性相对优秀，基本能够实现在 15 FPS 以上的识别速度。同时，界面也十分直观，能够看出各工件的边框效果以及数目列表。对轻微粘连的情况也有一定的鲁棒性。当然，也可以通过 Thresh 的调节滑块对边缘的选取进行微调，使结果更加理想。

综上，我们认为我们的实验结果是十分理想的。

六、体会和建议

6.1 体会

房江祎体会

本人主要任务为界面设计&SDK 接口设计，不变矩分类器设计，确定源程序设计框架与设计方法与原则等等。

在程序设计过程中，我曾经有过方向偏离，也曾经出现过技术难关无法克服，两人对接合作不符，甚至还因为设计方法不妥当导致程序混乱无法继续。但最终我克服困难，对代码进行重构以理清思路；利用正确的数据结构(队列)来克服困难；采取正确解耦方式与合理分工以正确对接；确定正确的设计模式(MVC)与设计方法(FSM)；在合作项目过程中，我加深了对项目规划，数据结构，任务分工，设计框架等多个方面的认识。

韦思成体会

在这次实验中，我负责的部分是图像处理环节的所有算法及优化，以及将分类器环节的算法耦合进图像处理环节，达到加速流程的效果。

在对灰度图像的规范化、连通域求解算法的不断优化的过程中，我对算法效率、时间空间复杂度的理解得到了加深。同时，由于使用了大量的静态的寄存数组，我也必须对它们的长度进行一定的调整，以实现较好完成任务，而不会有太多空间浪费的目标。

最终，我在图像处理环节单个环节的运算速度基本可以控制在 30ms 左右，与 OpenCV 的 cv::connectedComponentswithStats 的算法效率相当接近，这也说明我的算法优化基本是令人满意的。

6.2 建议

1、可以将 MFC 框架更改为更新的 WPF、Qt 或 UWP，这样能够查阅的资料更加充足，更方便进行快捷高效的界面设计。

2、建议可以使用 Github 提交项目，让大家熟悉项目实践的流程。

3、可以增加一些加分项，例如不使用背光就能完成工件分类等。

4、工件分类和数豆子两题太过相似，可以删去其中一题，换一道其他类型的题目，例如用 Haar 方法识别照片中的人脸等。

七、参考文献

OpenCV 文档：https://docs.opencv.org/4.5.5/
Microsoft Developer Network：https://docs.microsoft.com/en-ca/
章毓晋. (2017). 计算机视觉教程. Ren min you dian chu ban she.

指导教师的评语及评分	
综合评分	