中山大學
SUN YAT-SEN UNIVERSITY

《计算机视觉实验报告》

（实验五）

学院名称	：	数据科学与计算机学院
专业班级	：	17 级软件工程 5 班
学生姓名	：	张淇
学　　号	：	17343153
时　　间	：	2019 年 11 月 10 日

成绩 ：

实验五 ： 数字、括号的分割与识别

一、 实验内容

1. 任务一： 在作业 4 第二步完成的基础上，切割出红绿蓝这些区域内的数字(分别用不同颜色框标记)，并识别这些数字(可以用 OpenCV 已经训练好的模型)。
2. 任务二： 分割并识别水平括号(包括水平括号下边的数字和左右两端刻度值)。

二、 实验原理

1. 寻找连通块的Two-pass法

   （hw4的实验报告中已经介绍了此处不再赘述）

2. 数字识别的KNN算法：

kNN算法的核心思想是如果一个样本在特征空间中的k个最相邻的样本中的大多数属于某一个类别，则该样本也属于这个类别，并具有这个类别上样本的特性。该方法在确定分类决策上只依据最邻近的一个或者几个样本的类别来决定待分样本所属的类别。 kNN方法在类别决策时，只与极少量的相邻样本有关。由于kNN方法主要靠周围有限的邻近的样本，而不是靠判别类域

的方法来确定所属类别的，因此对于类域的交叉或重叠较多的待分样本集来说，kNN方法较其他方法更为适合。

三、 实验器材

Ubuntu 18.04， C++11, Opencv 3.2, Cimg

四、 实验过程与结果

（注：实验结果截图可以在 ans_img 文件夹下找到）

(一) 任务一

1. 找出图像中的数字、小数点

程序中对应的函数为：

```
vector<Rectan> find_nums_dots(CImg<uchar> img, CImg<uchar> &output, vector<Rectan> &nums_loc, vector<Rectan> &dots_loc);
```

这个功能与 HW4 中的任务二是除了范围（HW4 只需要坐标轴附近的数字，HW5 需要整张图的数字）之外基本一致，所以代码基本是拷贝 HW4 的。但是因为三张图片的数字各有特点（连通块长度、大小、长宽比，连通块长度）在判断“连通块是否为数字”这一个步骤中添加了关于上述特点的限制条件，于此同时也给了相关的条件进行判断连通块是否为小数点。如 ans.hpp 中的宏定义：

```
#define MIN_NUMBER_PIXELS 5
#define MAX_NUMBER_PIXELS 300
#define MIN_NUMBER_WIDTH 6
#define MAX_NUMBER_WIDTH 25
#define MIN_NUMBER_HEIGHT 6
#define MAX_NUMBER_HEIGHT 25
```

main.cpp/find_nums_dots()中的判断条件：

```
//找到包围连通块的最小矩形，用于确定该图形的位置。
if(run_vector[i].row < pix_top) pix_top = run_vector[i].row;
if(run_vector[i].row > pix_down) pix_down = run_vector[i].row;
if(run_vector[i].start_col < pix_left) pix_left = run_vector[i].start_col;
if(run_vector[i].end_col > pix_right) pix_right = run_vector[i].end_col;
//求连通块的长度
length += run_vector[i].end_col - run_vector[i].start_col + 1;

//curr_id != next_id，意味着遍历到下一个连通块
if (run_vector[i].id != run_vector[i+1].id){
    //数字
    if(length <= MAX_NUMBER_PIXELS && length >= MIN_NUMBER_PIXELS
        && pix_right - pix_left < MAX_NUMBER_WIDTH && pix_right - pix_left > MIN_NUMBER_WIDTH
        && pix_down - pix_top < MAX_NUMBER_HEIGHT && pix_down - pix_top > MIN_NUMBER_HEIGHT){
        needed_rectan.push_back(Rectan(pix_left,pix_right,pix_top,pix_down));
        nums_loc.push_back(Rectan(pix_left,pix_right,pix_top,pix_down));
    }
    // 小数点
    else if(length <= MAX_NUMBER_PIXELS && length >= MIN_NUMBER_PIXELS
            && pix_right - pix_left <= MIN_NUMBER_WIDTH && pix_down - pix_top <= MIN_NUMBER_HEIGHT
            && abs((pix_right - pix_left) - (pix_down - pix_top)) < 3){
                needed_rectan.push_back(Rectan(pix_left,pix_right,pix_top,pix_down));
                dots_loc.push_back(Rectan(pix_left,pix_right,pix_top,pix_down));
            }
    else{
        pass_id.push_back(run_vector[i].id);
    }
    length = 0, pix_left = 100000, pix_right = -1, pix_top = 10000, pix_down = -1;
```

然后我们就可以把找到的符合条件的连通块存到 vector<Rectan>中等待下一步操作（Rectan 是我定义的结构体，记录了能够容纳指定连通块的最小矩形的四个点的坐标）。

但是这一步骤中并没有办法保证满足上述约束条件的连通块都是数字或小数点，还会有一些“杂质”，这个是难以避免的（实验结果见 3）。

2. 识别图像中的数字

程序中对应的函数为：

```
Mat recognize_numbers(vector<Rectan> num_loc, vector<Rectan> dots_loc);
```

这一步骤主要使用的 KNN，将上述操作找到的数字和小数点从源图片中“截下来”，挑选部分比较有代表性的进行标记、做成 labeled_data，然后将他们作为样本进行训练。将训练得到的分类器再对上述所有满足“数字条件”的连通块进行判断，将判断的结果直接作为 text 打印到图片中与数字相邻的位置，小数点也打印到相邻的位置。得到的结果如下：

（注：上文已经提到，在“连通块是否为数字”的判断中无法剔除一些无法用约束条件限制的“杂质”，因此在下面的结果中也对其进行了识别，得到的结果也打印到图上，但不影响关键数据的识别）

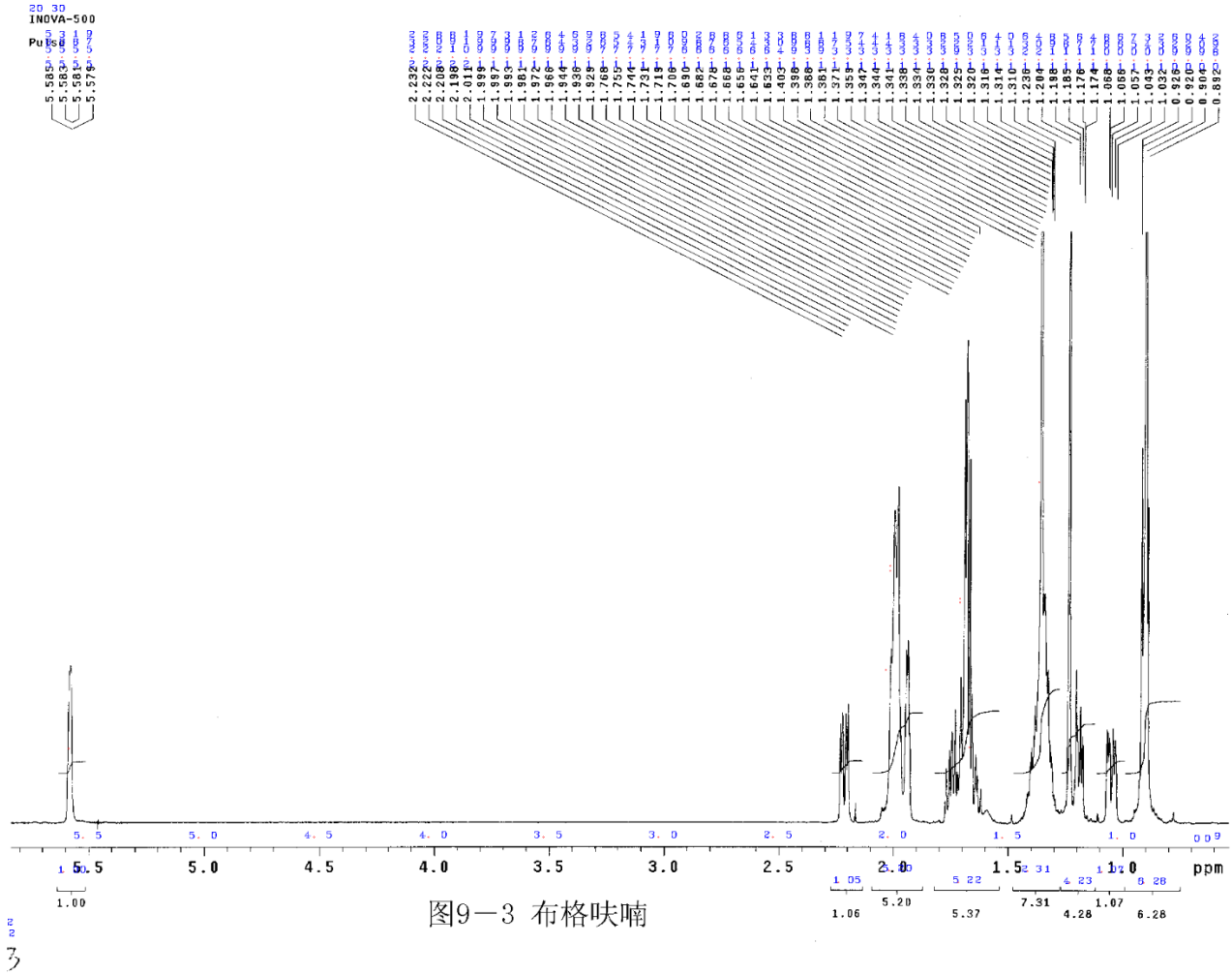
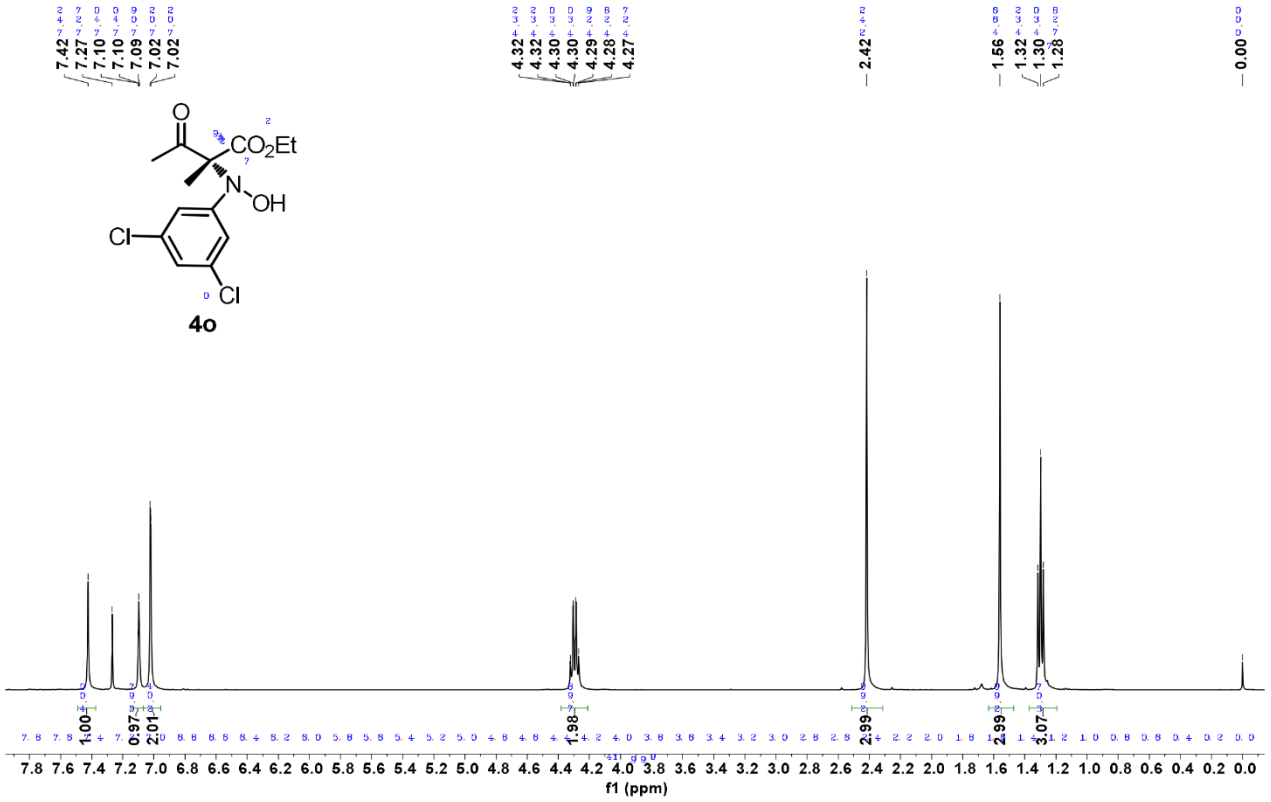
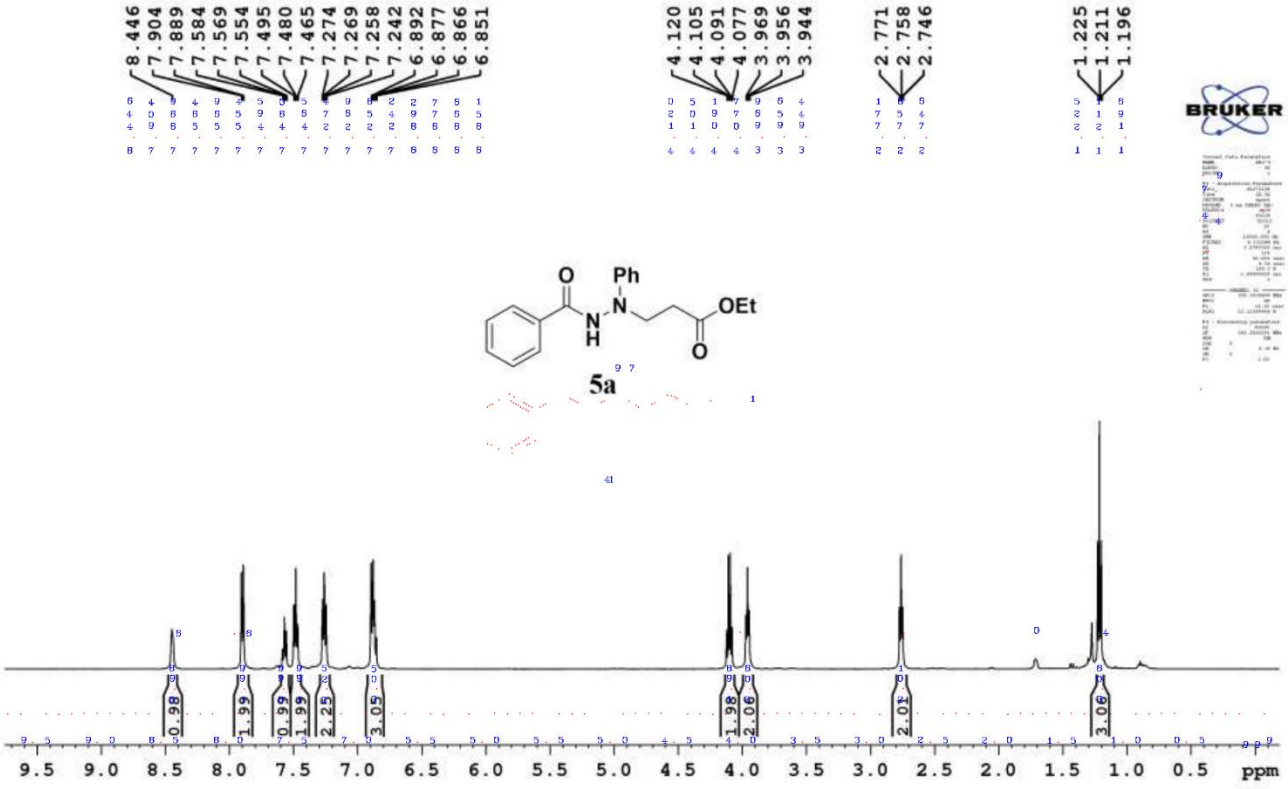


图9—3 布格呋喃





因为作业要求需要“能够对另外两张图进行同样的处理”，所以在程序中我**只训练了一个分类器**（训练样本横竖、大小、宽窄都存在差异），导致的结

果就是三张图中个别的比较“特殊”的数字（指的是与其他同类的其他数字差别较大的）被误判，但是整体识别的正确率在 98%左右，属于可以接受的范围。

3. 找到数字“聚集的区域”

程序中对应的函数为：

```
vector<vector<Rectan>> cut(vector<Rectan>needed_Rectan,vector<Rectan> &range,
 CImg<uchar> &output);
```

因为在步骤一中已经找到了所有属于数字和小数点的位置（找连通块使用的 two-pass 方法是遍历行的，所以找到的连通块也是按照最顶一行的行号进行排序的），所以可以遍历上述数字/小数点来找到处于相邻几行的其他数字，从而不断更新区域的上下左右的范围，直至遍历的下一个数字/小数点与区域的行数相差较大时停止，重复此操作来找到图片中所有的“区域”，然后再加上一些判断条件来剔除步骤一中得到的“杂质”产生的“区域”即可。

得到的结果如下图（步骤一也是下图）：

（注：在实验结果图中，将识别到的数字/小数点，包括“杂质”使用红色矩形将其包围，对于“区域”则是使用的红/绿/蓝的大框将其包围。如果在实验报告中的图看不清，则可以在 ans_img 文件夹中找到单独的图。）

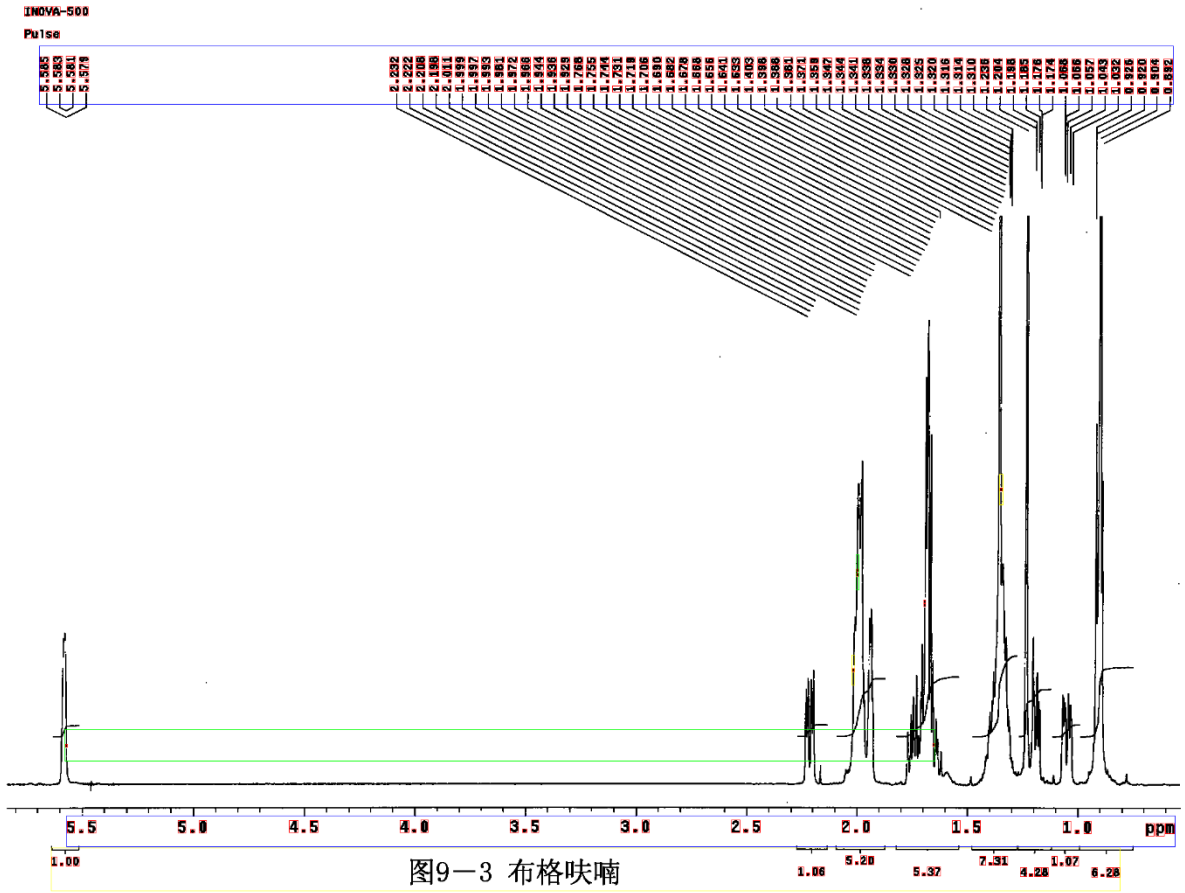
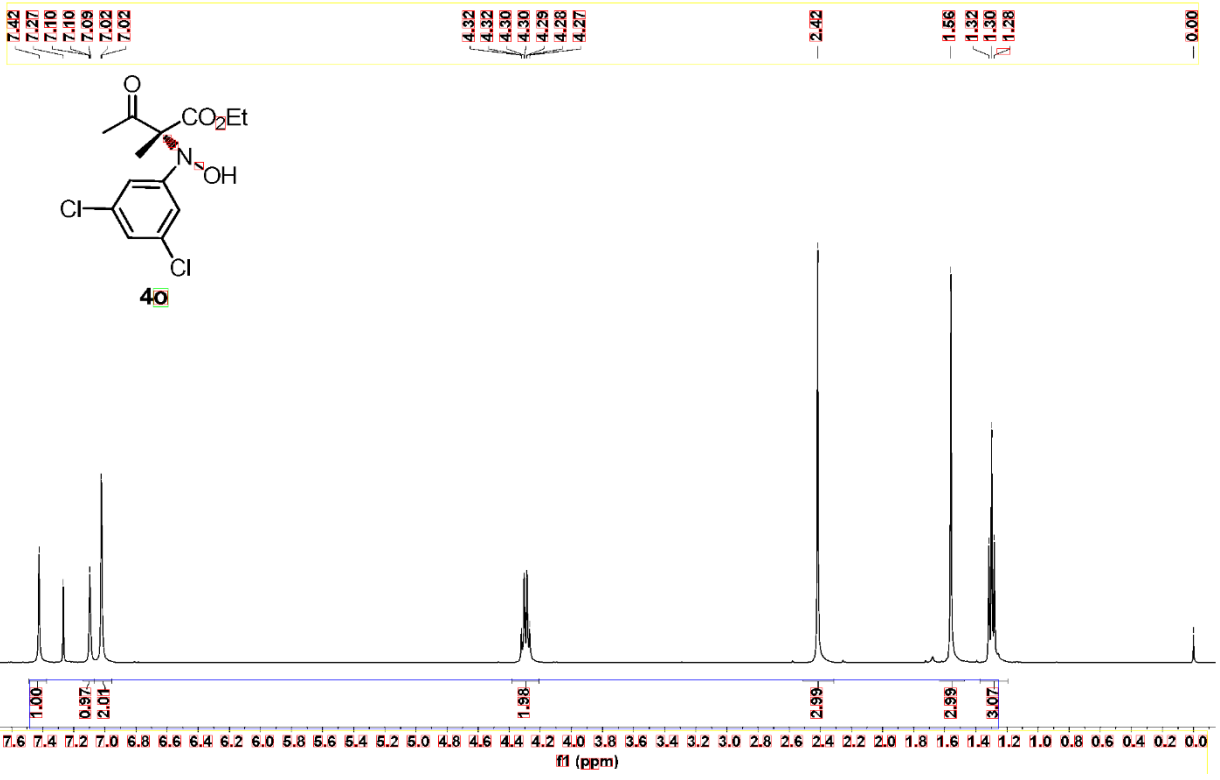
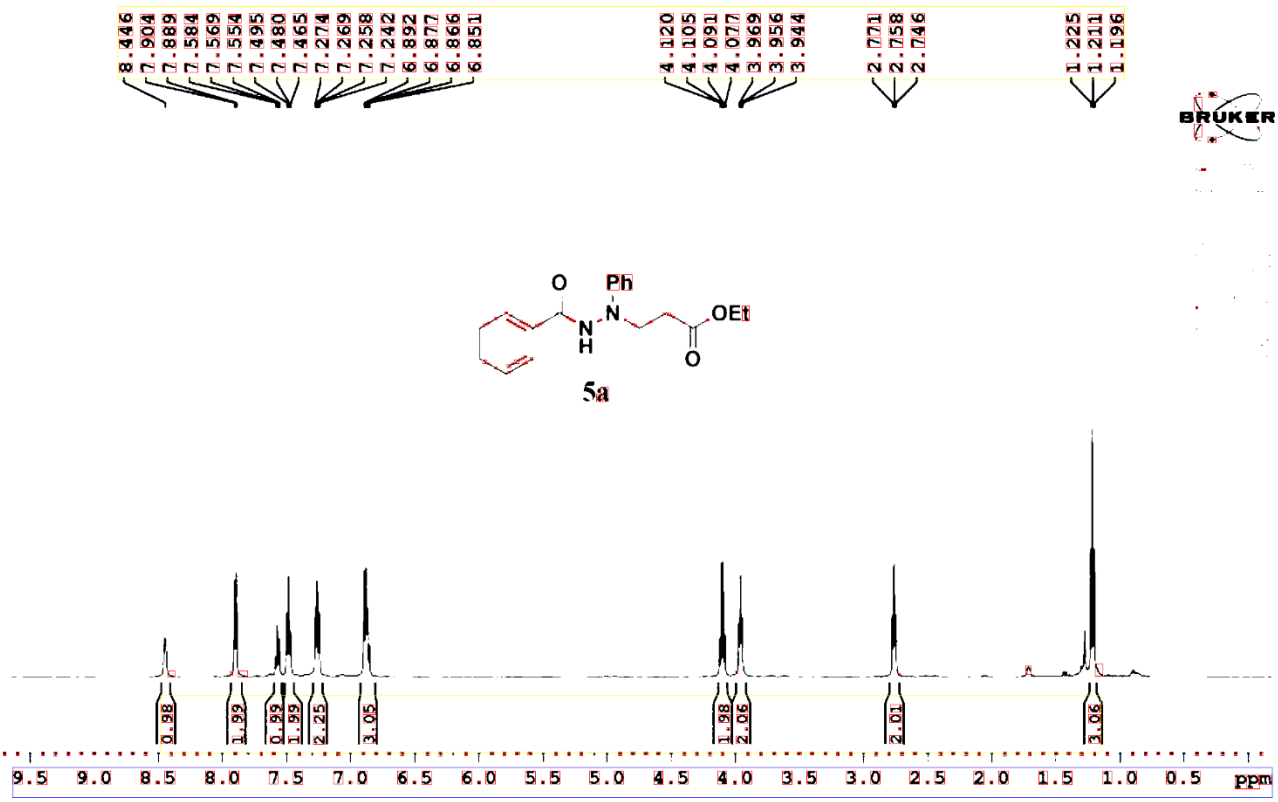


图9－3 布格呋喃



4o



(二) 任务二

程序中对应的函数为：

```
void bracket(CImg<uchar> img);
```

1. 识别括号

因为这个任务需要做的是在坐标轴附近找到括号，所以我们可以先找到坐标轴的位置——遍历整张图，拥有最多“黑块”的那一行就是坐标轴…

因为括号的在二值图像中也是连通块，所以也可以根据 two-pass 方法加上一些条件约束找到括号。在找括号这一个步骤中，我并没有遍历全图，而是以坐标轴的那一行为基准向两边找。这样做的好处有两个：一是节约时间，而是减少像任务一中“杂质”带来的影响，方便后续的操作。同样地，我们将括号用 vector<Rectan>存起来。

2. 过括号的“边界”向坐标轴引垂线

按照上述步骤操作完成之后就得到了括号的在图像中的具体位置，然后使用 Cimg 的 draw_line()函数向坐标轴引一条过括号“边界”的垂线，并且在交点位置使用 draw_circle()函数画一个圆来突出这个位置。

3. 按照“比例尺”求坐标

可以轻松的得到上述交点的位置，最后一步就是计算坐标。我用的方法很简单，就是找到坐标轴两个边界的坐标(x1,y)和(x2,y)，并且从“意义上”可以显然得到相对应的分成了 n 格，一个格对应的数值 v，再找一个基准点(x0,y), 基准点的数值为 v0。

那么这个交点的坐标对应的数值就是:

$$v0 - v * \frac{x - x0}{\frac{|x1 - y1|}{n}}$$

最后再将其转化为 string/char*类型作为 text 打印到源图像的合适位置即可。实验结果如下:

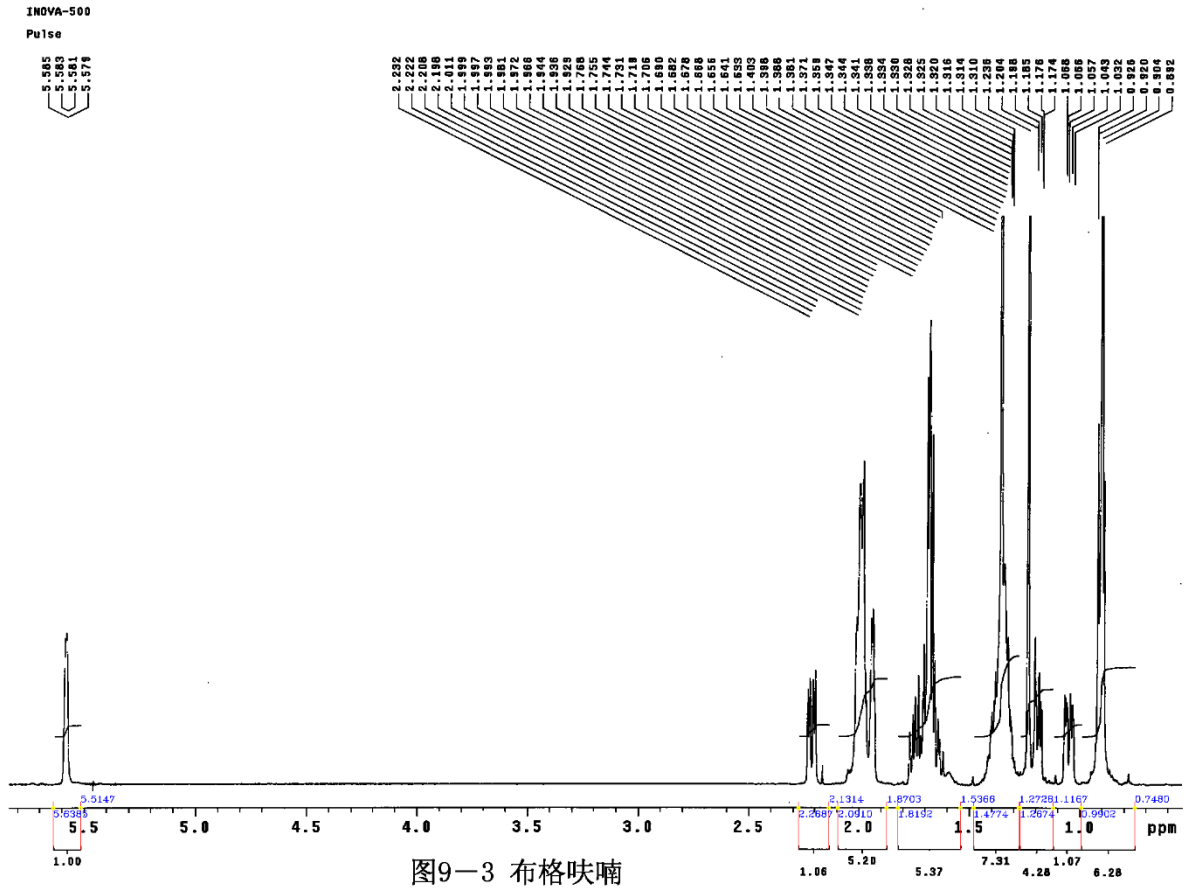
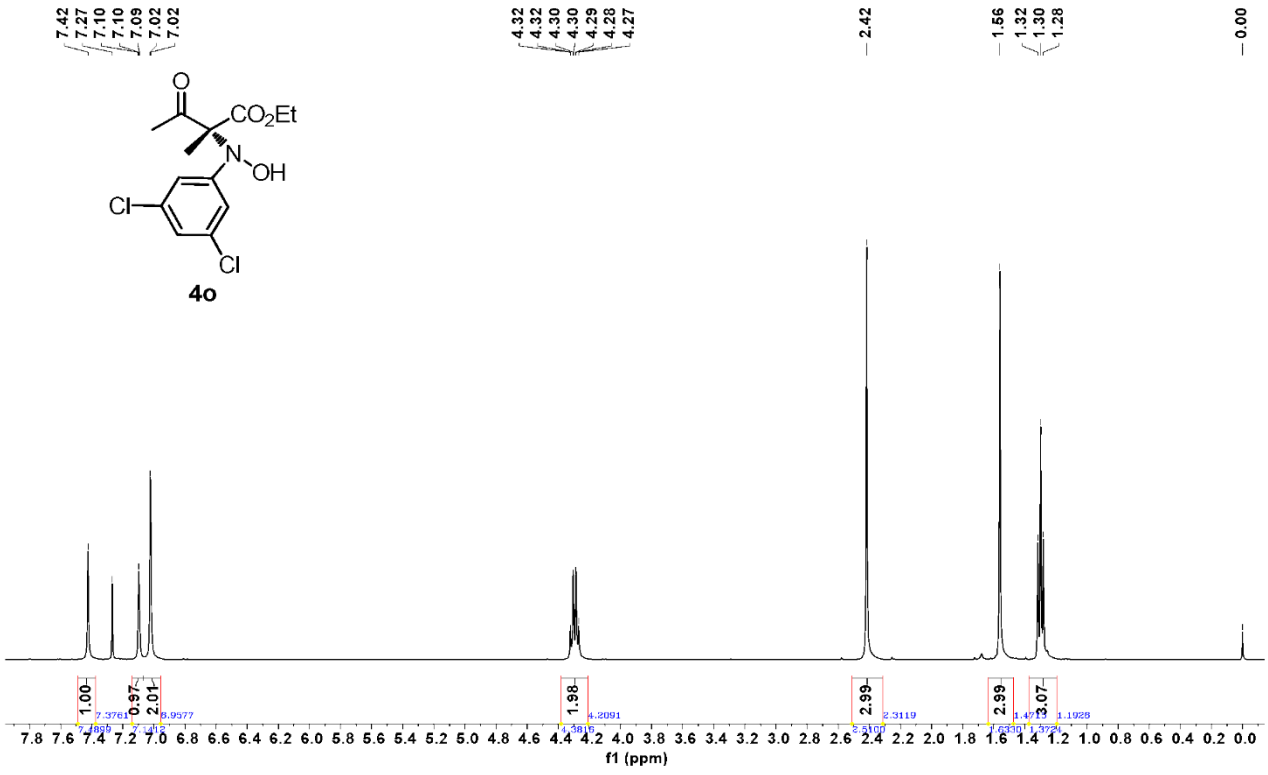
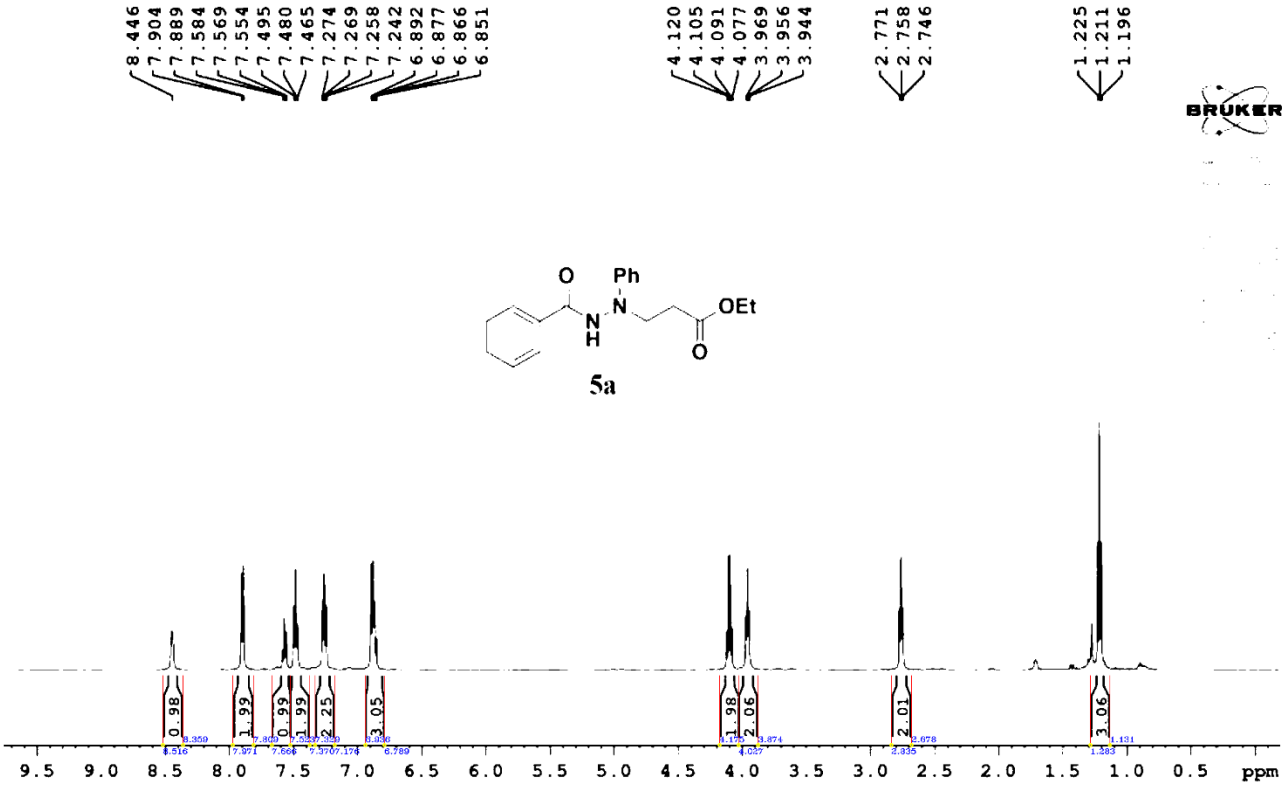


图9－3 布格呋喃

3




8.446
7.904
7.889
7.584
7.569
7.554
7.495
7.480
7.465
7.274
7.269
7.258
7.242
6.892
6.877
6.866
6.851
4.120
4.105
4.091
4.077
3.969
3.956
3.944
2.771
2.758
2.746
1.225
1.211
1.196
BRUKER
O
Ph
N
OEt
N
H
O
5a
0.98
1.99
0.99
1.99
2.25
3.05
1.98
2.06
2.01
3.06
9.5
9.0
8.5
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
ppm