西安科技大学

《软件建模与实践》

实验报告

题　　目　基于 ObjectARX 的 AutoCAD 二次开发锚杆支护系统的设计及自动绘图软件

院、系(部)　计算机科学与技术

专业及班级　计算机科学与技术 1901 班

学　　号　19208049010

姓　　名　赵 琦

日　　期　2020 年 1 月 7 日

1 题目要求

锚杆支护是指在边坡、岩土深基坑等地表工程及隧道、采场等地下硐室施工中采用的一种加固支护方式。AutoCAD 是 Autodesk 公司首次于 1982 年开发的自动计算机辅助设计软件，用于二维绘图、详细绘制、设计文档和基本三维设计，现已经成为国际上广为流行的绘图工具。要求设计一个软件，实现根据不同的计算方法对矿井巷道参数进行设计，并调用 AutoCAD 绘制巷道锚杆锚索支护设计图。具体实验要求如下：

（1）对系统进行架构设计、详细的需求分析设计，设计需求分析的问题，并给出需求分析设计报告；

（2）画出系统的用例图、类图、交互图、活动图等；

（3）要求在客户端对巷道参数进行输入和计算，并能调用 AutoCAD 根据设计的参数进行绘制巷道锚杆（索）支护图。

（4）设计并实现该系统，要求系统具有一定的可拓展性和良好的用户交互性，例如计算方法的增加、具有不同巷道的绘制方法，对输入的数据进行校验等。

2 需求说明

根据以上题目要求分析，该锚杆支护系统需要具备以下几个功能模块：

1. 工程管理模块：完成工程的新建、打开和保存功能。每个子功能应具有良好的用户交互设计；

2. 巷道参数管理模块：完成对巷道参数的输入及数据验证功能；

3. 计算方法模块：根据不同的计算方法，输入相应的参数进行计算；

4. 自动绘图模块：根据巷道参数、锚杆、锚索参数绘制不同的 AutoCAD 设计图。

3 总体设计

3.1 架构设计

对软件的功能需求进行分析，设计本系统的架构为一下三种子模块：

1. 负责与用户交互、处理数据的可执行文件 MFCad.exe；

2. 负责绘图的 AutoCAD 加载文件 INGPrjcadc.arx；

3. 桥接文件 bridge.ini，负责将以上两程序的数据进行桥接，完成参数的传递。

本系统的软件架构图如下图所示：

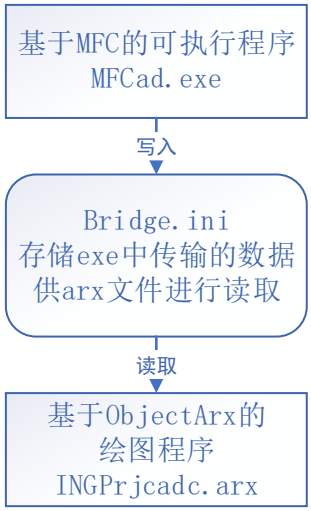

图 1 系统架构图

3.2 业务逻辑设计

分析软件各模块功能，设计出本系统的用例图如下图所示：

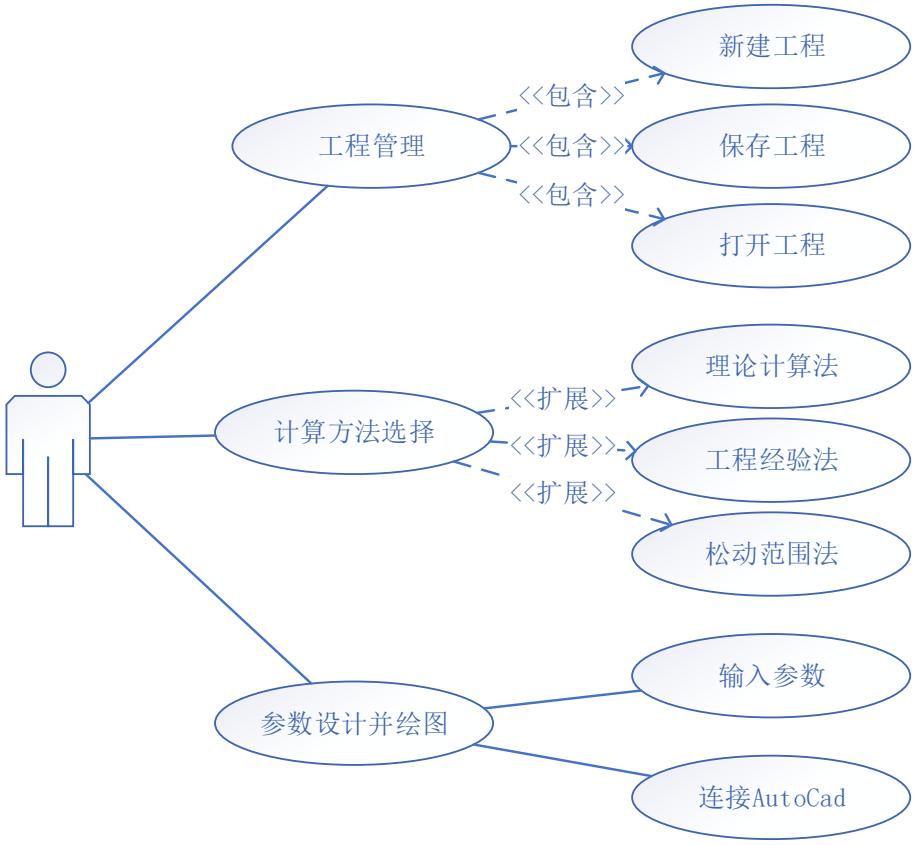

图 2 系统用例图

4 详细设计

4.1 工程管理模块

工程管理模块负责工程的新建、打开及保存，自然而然就需要实现对文件的保存、读取和打开功能，编写 FileUtils 工具类实现文件的创建、读写等原子函数，再编写 MFC 按钮的点击事件对其进行个性化调用。

由于同一时刻只能有一个工程打开，故采用单例的模式构建一个工程，与工程对应的界面ProjectDialog也是同理,实例化一个全局的ProjectDialog对象，确保加载的是只有一个工程。

```
CProjectDialog* pDlg = NULL;
```

打开工程的代码如下所示：

```
void CMFCadDlg::OnOpenProject()
{
    // TODO: 在此添加命令处理程序代码
    TCHAR szFilter[] = TEXT("参数文件(*.ini)|*.ini|所有文件(*.*)|*.*||");
    CFileDialog fileDlg(TRUE, TEXT("ini"), NULL, 0, szFilter, this);
    CString strFilePath;
    if (IDOK == fileDlg.DoModal())
    {
        strFilePath = fileDlg.GetPathName();
    }
    if (!strFilePath.IsEmpty()) {
        CArcProjectBuilder::GetInstance()->SetFileUrl(strFilePath);
        CArcProjectBuilder::GetInstance()->BuildAll();
        MessageBox(_T("文件打开成功！参数载入完毕"));
        CArcProjectBuilder::GetInstance()->SetSavedToFile(FALSE);
         pDlg = new CProjectDialog();
        pDlg->Create(IDD_PROJECT_DIALOG, this);
        pDlg->ShowWindow(SW_SHOW);
    }
}
```

新建工程分为三种不同的情况，若当前无工程打开，直接新建即可；若当前有工程打开，需要查看当前工程是否已经保存了，若已保存则直接新建，若未保存需要询问是否保存当前工程，或是不保存直接新建，同时需要对当前软件中所维护的信息进行修改。核心代码如下所示：

```
void CMFCadDlg::OnNewProject()
{
```

```
CArcProjectBuilder::GetInstance()->GetSavedToFile()
    if (CArcProjectBuilder::GetInstance()->GetSavedToFile() == TRUE
        && (!CArcProjectBuilder::GetInstance()->GetFileUrl().IsEmpty()))
    {
        // 打开的工程已经保存了
        CArcProjectBuilder::GetInstance()->SetFileUrl(_T(""));
        CArcProjectBuilder::GetInstance()->SetSavedToFile(FALSE);
        CArcProjectBuilder::GetInstance()->SetTunnelProject(new CTunnelProject());
        CArcProjectBuilder::GetInstance()->SetArcTunnel(new CArcTunnel());

        if (pDlg == NULL)
        {
            pDlg = new CProjectDialog();
            pDlg->Create(IDD_PROJECT_DIALOG, this);
        }
        pDlg->ShowWindow(SW_SHOW);
    }
    else if (CArcProjectBuilder::GetInstance()->GetSavedToFile() == TRUE &&
        CArcProjectBuilder::GetInstance()->GetFileUrl().IsEmpty()) {
        // 当前窗口是新建窗口
        CArcProjectBuilder::GetInstance()->SetFileUrl(_T(""));
        CArcProjectBuilder::GetInstance()->SetSavedToFile(FALSE);
        CArcProjectBuilder::GetInstance()->SetTunnelProject(new CTunnelProject());
        CArcProjectBuilder::GetInstance()->SetArcTunnel(new CArcTunnel());
        if (pDlg == NULL)
        {
            pDlg = new CProjectDialog();
            pDlg->Create(IDD_PROJECT_DIALOG, this);
            pDlg->ShowWindow(SW_SHOW);
        }
    }
    else if(CArcProjectBuilder::GetInstance()->GetSavedToFile() == FALSE)
    {
        // 打开的工程没保存
        if(IDYES== MessageBox(_T("当前工程尚未保存，是否直接新建？"), 0, MB_YESNO))
        {
            if (pDlg != NULL)
            {
                CArcProjectBuilder::GetInstance()->SetProjectSaveToInstance(TRUE);
                std::cout << "destory pdlg\n";
                pDlg->DestroyWindow();
                delete pDlg;
                pDlg = NULL;
                std::cout << "test destory pdlg\n";
```

```
                CArcProjectBuilder::GetInstance()->SetProjectSaveToInstance(FALSE);
            }
                CArcProjectBuilder::GetInstance()->SetFileUrl(_T(""));
                CArcProjectBuilder::GetInstance()->SetSavedToFile(FALSE);
                CArcProjectBuilder::GetInstance()->SetTunnelProject(new
CTunnelProject());
                CArcProjectBuilder::GetInstance()->SetArcTunnel(new CArcTunnel());
                if (pDlg == NULL)
                {
                    pDlg = new CProjectDialog();
                    pDlg->Create(IDD_PROJECT_DIALOG, this);
                    pDlg->ShowWindow(SW_SHOW);
                }
        }
    }
}
```

保存当前工程参数到文件，核心代码如下所示：

```
CArcProjectBuilder::GetInstance()->SetFileUrl(strFilePath);
if ((CArcProjectBuilder::GetInstance()->SaveProjectToFile() == true)
    && (CArcProjectBuilder::GetInstance()->SaveTunnelFlagToFile() == true)
    && (CArcProjectBuilder::GetInstance()->SaveParametersToFile() == true)
    && (CArcProjectBuilder::GetInstance()->SaveArcTunnelInfoToFile() == true)
    && (CArcProjectBuilder::GetInstance()->SaveMethodToFile() == true)
    ) {
    CArcProjectBuilder::GetInstance()->SetSavedToFile(TRUE);
    MessageBox(_T("保存工程信息成功"));
}
else {
    CArcProjectBuilder::GetInstance()->SetSavedToFile(FALSE);
    MessageBox(_T("保存工程信息失败，参数可能不完善"));
}
```

4.2 巷道参数管理模块

对打开的文件进行读写处理，从工程构造 ProjectBuilder 类实现对本系统中一个工程的参数的构建和保存等功能，利用建造者模式创建工程类这一复杂对象，工程类包括基本的巷道属性、支护方式、计算方法及各锚杆锚索参数等。

对项目中所需要的参数进行分析，设计出如下图所示的类间关系图，将巷道的公共参数封装在 CTunnel 类，工程信息如设计者、图纸编号等信息抽象为 CTunnelProject 类，相应的锚杆锚索信息也进行封装，在建造者类中的成员方

法，通过 fileUrl 这一变量控制本地文件，从而实现数据的初始化、保存、加载等功能。

图 3 锚杆支护系统 UML 类图

4.3 计算方法模块

每个工程可以选择相应的计算方法对输入的参数进行计算，得到锚杆、锚索的值，将不同的计算方法采用工厂模式进行封装，在 BuildMethod 方法中对工程的各个计算方法进行初始化，采用静态转型方法进行向下转型，便于工程中抽象得调用。

```
class CMethod { ... };
class CTheoryCalMethod { ... };
class CProExpMethod { ... };
class CLooseRangeMethod { ... };
class CMethodFactory { ... };
class CTheroyMethodFactory { ... };
class CProExpMethodFactory { ... };
class CLooseRangeMethodFactory { ... };
```

图 4 计算方法类结构

对计算方法进行初始化的核心代码如下所示：

```
factory = new CTheroyMethodFactory();
method = factory->createMethod();
// 静态转型
theory = static_cast<CTheoryCalMethod *>(method);
```

4.4 自动绘图模块

要调用 AutoCAD 绘图，首先通过注册表获取到 AucoCAD 的安装路径，通过创建进程的方式启动 AutoCAD，并令其在启动时，自动加载编写好的 INGPrjcadc.arx 文件绘图。通过注册表获取 AutoCAD 安装路径的代码如下：

```
CString CFileUtil::GetAppRegeditPath()
{
    CString strAppName("SOFTWARE\\Autodesk\\AutoCAD\\R21.0\\ACAD-0001\\Install");
    std::cout << strAppName.GetString() << std::endl;
    HKEY hKey;
    CString strAppRegeditPath("");
    TCHAR szProductType[MAX_PATH];
    memset(szProductType, 0, sizeof(szProductType));
    DWORD dwBuflen = MAX_PATH;
    LONG lRet = 0;
    // 打开注册表，只有打开后才能进行其他操作
    lRet = RegOpenKeyEx(HKEY_LOCAL_MACHINE, //要打开的根键
        LPCTSTR(strAppName), // 要打开的子键
        0, //这个一定为0
        KEY_WOW64_64KEY | KEY_QUERY_VALUE,//指定打开方式为读
        &hKey
    );
    if (lRet != ERROR_SUCCESS)
    {
        printf("open error!\n");
        return strAppRegeditPath;
    }
    else
    {
        // 下面开始查询
        lRet = RegQueryValueEx(hKey, //打开注册表时返回的句柄
            TEXT("INSTALLDIR"), //要查询的名称
            NULL, NULL,
            (LPBYTE)szProductType,
            &dwBuflen);
    }
    if (lRet != ERROR_SUCCESS)
```

```
    {
        printf("read error!\n");
        return strAppRegeditPath;
    }
    else
    {
        RegCloseKey(hKey);
        strAppRegeditPath = szProductType;
    }
    std::cout << strAppRegeditPath.GetString() << std::endl;
    return strAppRegeditPath;
}
```

负责自动绘图的 Arx 文件通过调用 ObjectArx 提供的各种绘图 Api，将绘制直线、矩形、拱形等基本函数封装起来，在不同种巷道的绘制中进行调用即可。

将绘制的图形添加到模型空间，需要封装一个关键的函数：

```
//将实体添加到模型空间
AcDbObjectId CDrawUtil::PostToModelSpace(AcDbEntity * pEnt)
{
    //检查输入参数的有效性
    assert(pEnt);
    //获得当前图形数据库的块表
    AcDbBlockTable *pBlkTbl = NULL;
    acdbHostApplicationServices()->workingDatabase()->getBlockTable(pBlkTbl,
AcDb::kForRead);
    //获得模型空间对应的块表记录
    AcDbBlockTableRecord *pBLkTblRcd = NULL;
    pBlkTbl->getAt(ACDB_MODEL_SPACE, pBLkTblRcd, AcDb::kForWrite);
    pBlkTbl->close();
    //将实体添加到模型空间的块表记录
    AcDbObjectId entId;
    Acad::ErrorStatus errorStatus = pBLkTblRcd->appendAcDbEntity(entId, pEnt);
    if (errorStatus != Acad::eOk)
    {
        pBLkTblRcd->close();
        delete pEnt;
        pEnt = NULL;
        return AcDbObjectId::kNull;
    }
    //关闭模型空间表记录和实体
    pBLkTblRcd->close();
    pEnt->close();
    return entId;
    }
```

对三种巷道的绘制方式进行分析，不同巷道的绘制方法不同，设计出绘制函数如下所示，在具体的每个巷道中根据彼此的差异分别设计不同的绘制函数。

```
virtual void DrawProject(); // 绘制工程信息的表格
//绘制断面图形，三种巷道绘制断面方式各不相同
virtual void DrawTunnel();
//顶部锚杆托梁
virtual void DrawTopTuoLiang(CBolt bolt);
// 绘制顶部锚杆
virtual void DrawTopBolt(CBolt bolt);
//绘制顶视图的网格线
virtual void DrawTopViewNet(CBolt bolt);
virtual void DrawLeftBolt(CBolt bolt);
virtual void DrawLeftTuoLiang(CBolt bolt);
virtual void DrawLeftViewNet(CBolt bolt);
virtual void DrawRightBolt(CBolt bolt);
virtual void DrawRightTuoLiang(CBolt bolt);
virtual void DrawRightViewNet(CBolt bolt);
// 绘制锚索
virtual void DrawCable(CCable cable);
```

5 测试与实现

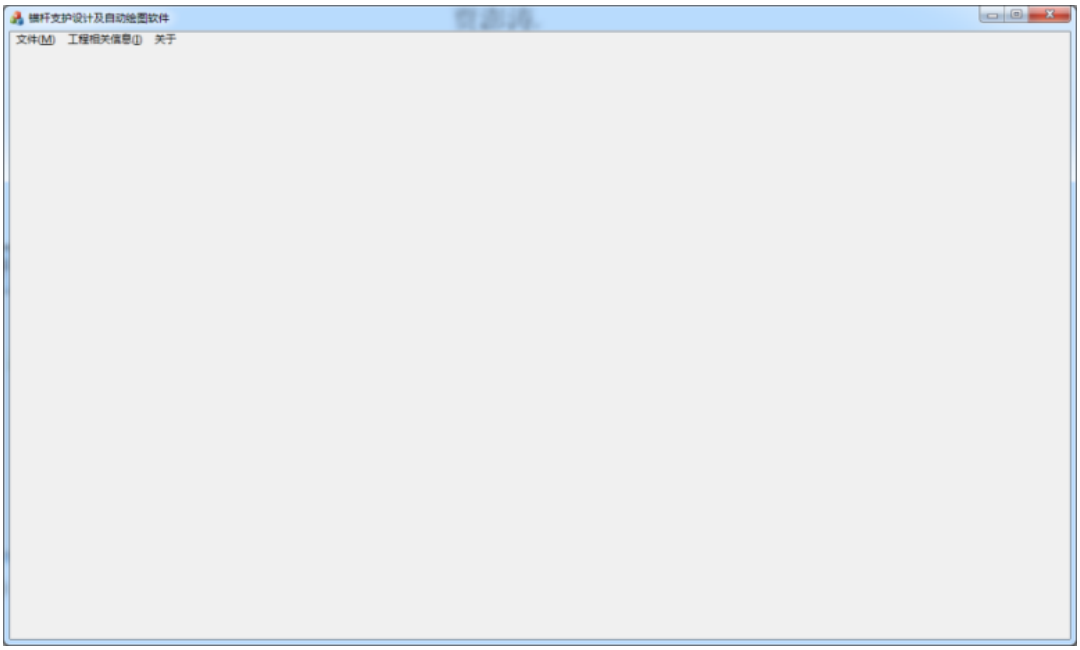

图 5 软件主界面

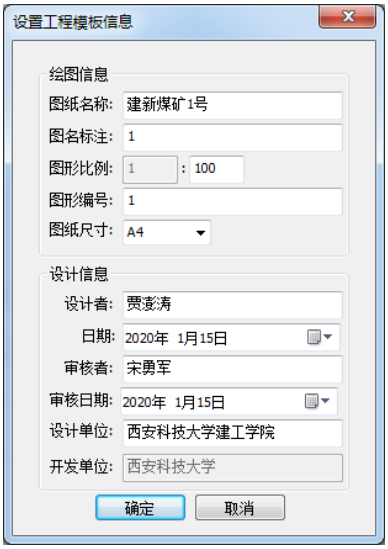

图 6 新建工程界面

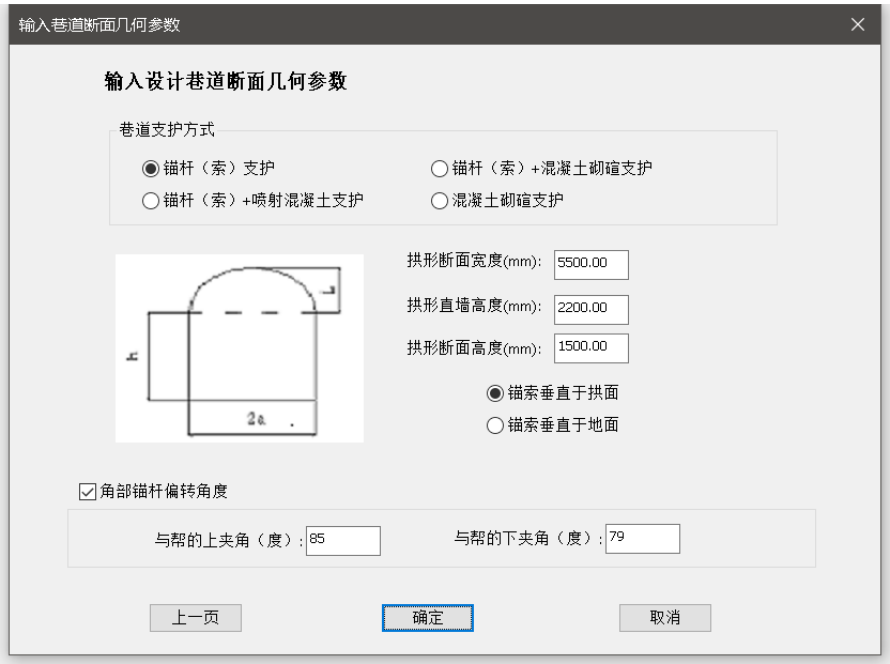

图 7 巷道参数输入界面

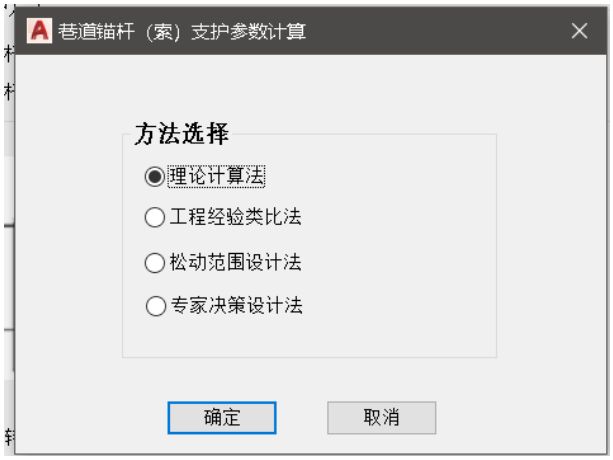

图 8 方法选择界面

图 9 专家决策设计法界面

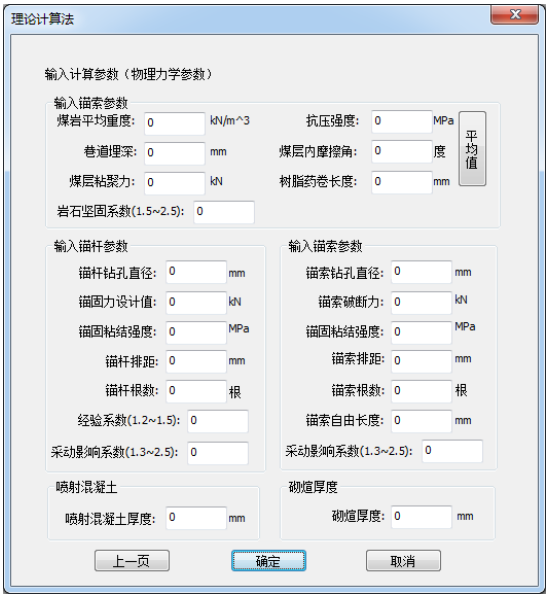

图 10 理论计算法界面

图 11 工程经验类比法

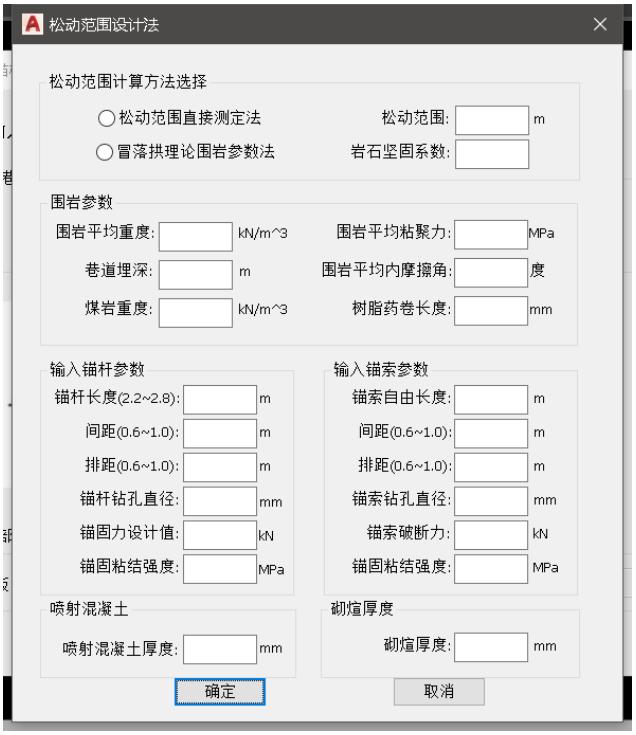

图 12 松动范围设计法界面

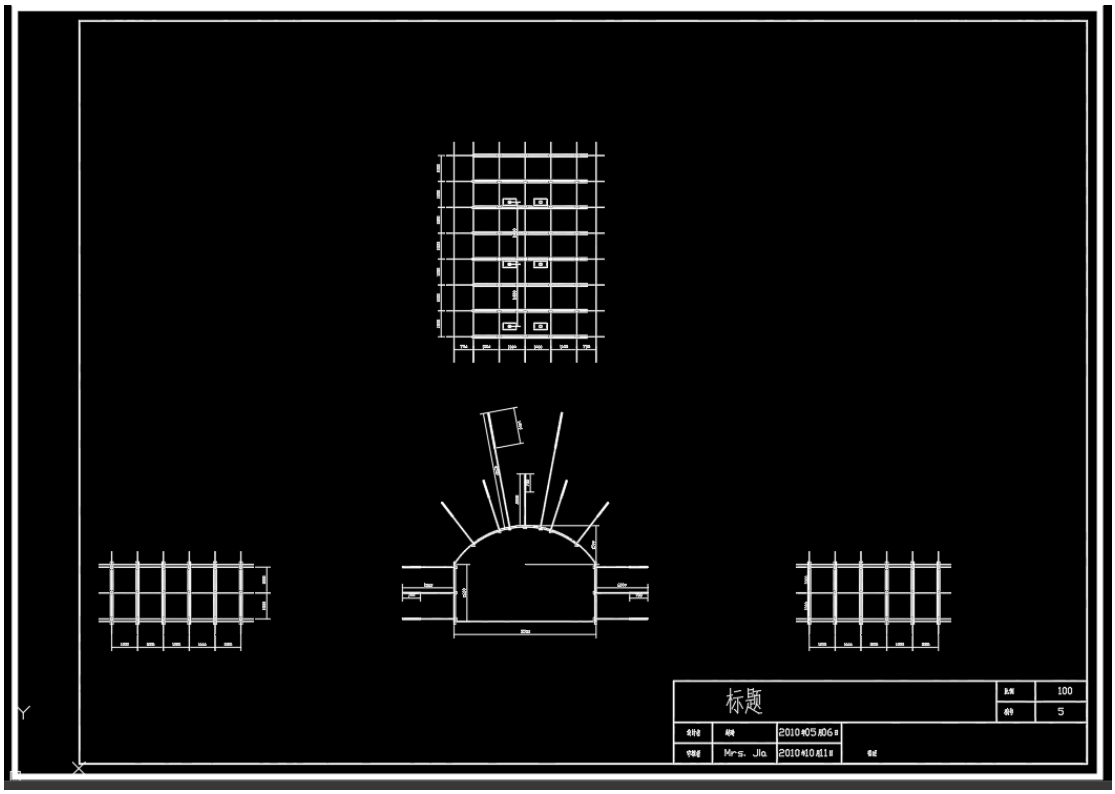

图 13 绘制锚杆支护设计图总览

图 14 工程信息标注表格

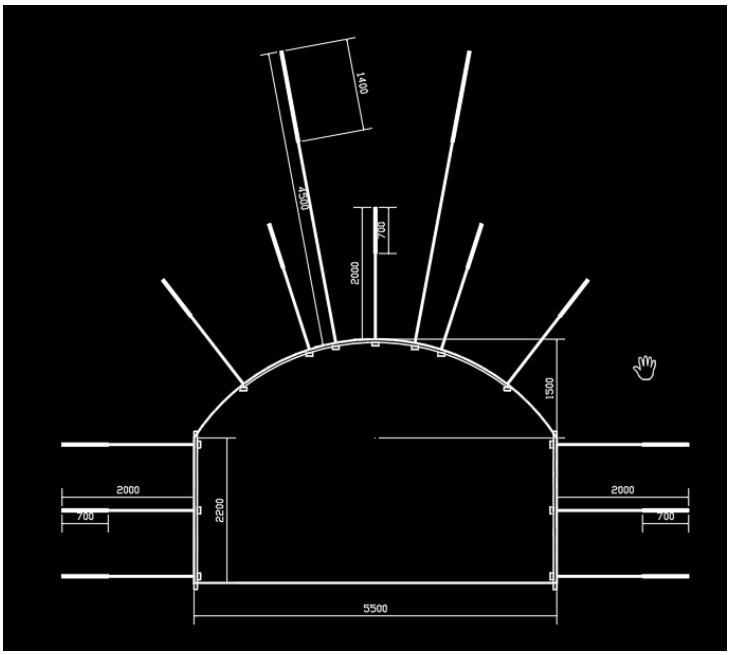

图 15 巷道断面图

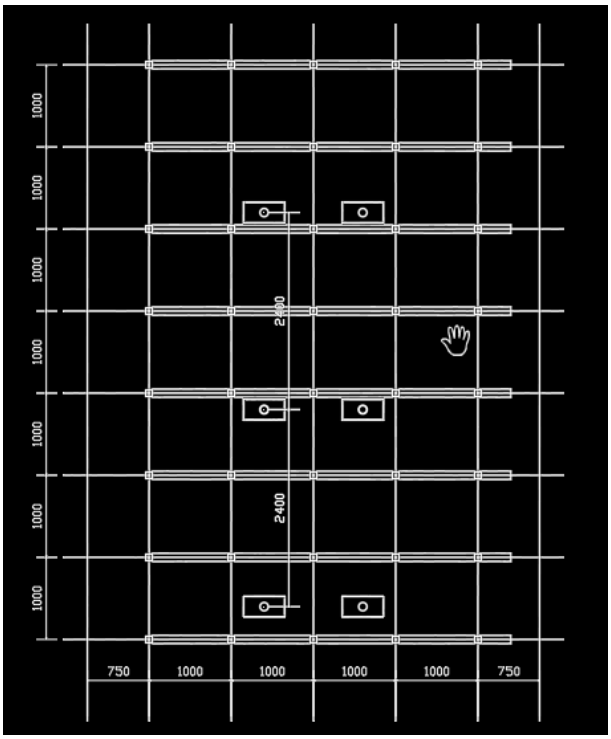

图 16 顶视图

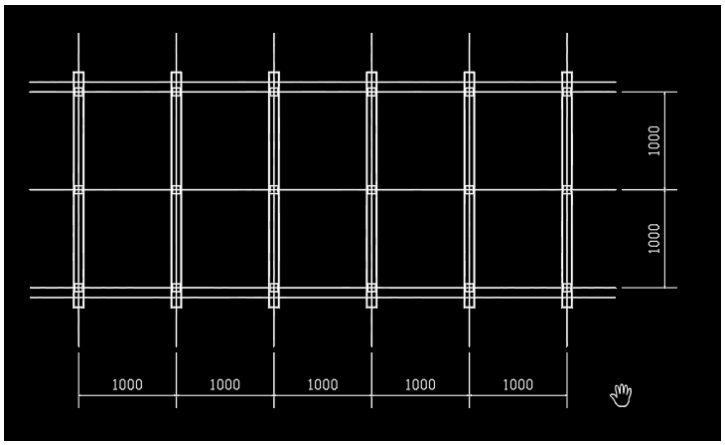

图 17 左视图（右视图与其类似）

6 总结

第一次编程实际应用中的软件，更是与之前所编的软件相差较大的 AutoCAD 自动绘图工具，从一开始了解自己要用什么技术、什么工具，到后面独立设计项目架构，子模块间的结构，市面上关于 ObjectARX 的 AutoCAD 学习资料甚少，所以一开始处处碰壁，在数次踩坑后最终还是开发出一个基本能用的软件，历时三个月，自己对于 C++和 MFC 的应用更加熟练了，ObjectArx 开发实现了从 0 到 1 的质的飞跃，也感觉到自己更好地成为了一个问题解决者。

在此过程中，我更加明白了需求沟通的重要性，贾老师总是能站在用户的角度提出很多的建议，包括测试文档更是让我感受到软件开发流程中的严谨性。关于开发日志和测试日志，我有时候会写，但大多数时间一改 bug，改完改不完的就到深夜了，也不知道该记录些什么东西，总是潦草几笔就盖过了，在这一部分自己还需要多努力。