目录

[1. 引言 2](#_Toc50883478)

[1.1 编写目的 2](#_Toc50883479)

[1.2 背景 2](#_Toc50883480)

[1.3 软件名称 2](#_Toc50883481)

[1.4 开发环境 2](#_Toc50883482)

[2. 总体设计 3](#_Toc50883483)

[2.1 需求概述 3](#_Toc50883484)

[2.2 系统结构 3](#_Toc50883485)

[3. 详细设计 3](#_Toc50883486)

[3.1 参数加载模块 3](#_Toc50883487)

[3.2 数据传输模块 5](#_Toc50883488)

[3.3 逻辑控制模块 9](#_Toc50883489)

[3.4 状态读写模块 9](#_Toc50883490)

[3.5 语言解析模块 9](#_Toc50883491)

[4. 关于程序的编译与运行 9](#_Toc50883492)

**设计说明书**

# 引言

## 1.1 编写目的

详细说明面向3C行业智能工业机器人控制器系统各个模块的功能以及实现原理，为机器人工程师提供开发基础。

## 1.2 背景

随着工业4.0的发展，3C行业机器人对多种控制方式、多传感器信息、多种工艺性提出了新的挑战。本软件基于WIN7系统的Intel处理器硬件平台，设计开发了一种基于RTX实时系统的工业机器人控制系统。

## 1.3 软件名称

面向3C行业智能工业机器人设计的控制系统V1.0

## 1.4 开发环境

开发硬件环境：Inrel Core i5-4570 CPU @3.20GHz，内存4G

开发软件环境：windows7 64位，Visual Studio 2017

# 总体设计

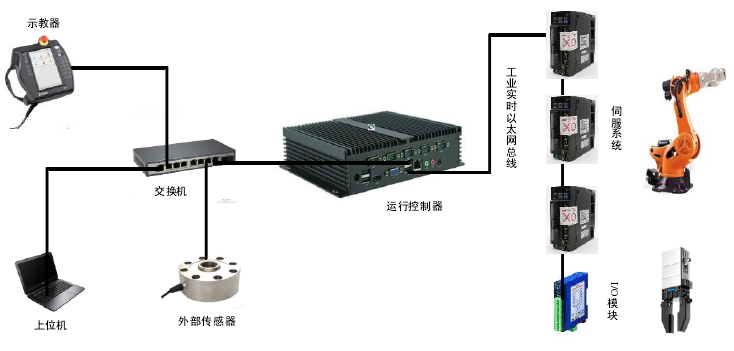
本设计的工业机器人控制系统硬件框架如图2.1。整个机器人控制器系统再硬件上主要由示教器、上位机、运行控制器、伺服系统及电机、I/O设备和外部传感器设备组成。运行控制器与伺服系统和I/O设备间采用工业实时以太网总线的形式进行交互，而示教器、上位机以及外部传感器与运行控制器之间采用Socket方式进行通信。

图2.1 机器人控制系统硬件框架

## 2.1 需求概述

机器人厂商通常提供两类人机交互平台：示教器和机器人集成开发环境。示教器侧重于在控制器运行过程中对机器人运动的控制、调试与监控；而机器人集成开发环境侧重于对机器人工程的编辑与配置、控制系统内用户任务的调试和机器人运动数据的离线调试与分析。本机器人控制系统可满足对控制器的快速开发及部署，服务于面向3C行业的智能工业机器人。

## 2.2 系统结构

如下图2.2所示为机器人控制系统的整体结构：

图2.2 机器人控制系统结构图

由上图所示为机器人控制系统结构图，可看出机器人控制系统主要由以下几个模块组成：

1. 参数加载模块

该模块的主要作用是为机器人控制器提供从RobotStudio中获取配置参数的接口。

1. 数据传输模块

该模块的主要作用是为机器人控制器提供和TeachView上位机进行数据交互的接口。

1. 逻辑控制模块

该模块主要作用是根据参数加载模块和数据传输模块所得到的的数据进行处理，根据不同的数据进行处理，调用相应的运动控制内核API，完成相应的运动控制功能。

1. 状态读写模块

该模块的主要作用是读取编码器数值，计算轴关节坐标系的位置，计算伺服电机的位置，计算工具坐标系相对世界坐标系的位置，并将这些数据传输到TeachView上位机示教软件中，使得上位机可以实时显示这些位置信息。

1. 语言解析模块

该模块的主要作用是对用户在上位机示教软件所编写的程序进行文本分析和语法分析，判断用户所编写的程序是否存在语义语法上的错误，若没有问题，程序讲进入执行阶段，根据不同的指令进行相应的运动学算法计算，调用相应的运动控制内核API完成对应的运动控制功能。

# 详细设计

## 3.1 参数加载模块

控制器在运行启动过程中，首先会加再一个XML文件，该文件配置好了机器人有关参数信息，机器人本体配置参数如下表格所示：

表3.1 机器人本体配置参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 配置属性名及参数接示 | 取值说明 |
| 机器人名称 | name:使用的机器人名称 | 无 |
| 机器人类型 | type:机器人类型 | 普通六轴机器人、其他 |
| 机器人D-H参数 | theta:运动关节角  d:机械臂连杆偏距  a:机械臂连杆长度  alpha:机械臂杆件  offset:关节偏距角 | 根据实际机器人本体参数说明并建立各关节确定  单位：度、毫米 |
| 机器人各轴极限参数 | maxPos:正向最大角度  minPos:负向最大角度  maxVel:负向最大角度  maxAcc:最大加速度  maxDec:最大加速度  maxJerk:最大加加速度 | 根据机器人电机性能和机械安装确定（由机器人厂商提供准确参数）  单位：度、毫米 |
| offset:关节零点相对伺服零点的偏移量  couping:各轴间的耦合关系  direction:各关节的方向 | 根据建立的机器人坐标系与伺服电机的零点和运动方向确定 |
| 机器人运动参数 | velAxis:轴运动速度  accAxis:轴运动加速度  jerkAxis:轴运动加加速度  velPath:路径速度  accPath:路径加速度  jerkPath:路径加速度  velOri:方向速度  accOri:方向加速度  jerkOri:方向加加速度 | 根据实际需要配置机器人运动时的最大速度、加速度、加加速度 |
| 机器人基座标系相对世界坐标系的位置 | base:基座标系  tool:工具坐标系 | 根据现场作业环境设置，当极坐标系与世界坐标系重合时，各坐标数值为0 |

机器人系统参数配置模块所得的配置文件固定名为RobotConfig.xml，这一文件是由另一个自主开发的系统软件胜过。通过使用TintXMl的类库，可以方便的生成所需的配置文件。RobotConfig.xml配置文件中每个机器人实例包括<BaseInfo>、<DH>、<Servo>、<Dynamic>和<Coordinate>节点，分别对应机器基本信息、机器人D-H参数、机器人各轴极限参数、机器人运动参数和机器人基坐标系相对世界坐标系的位置。生成的配置文件RobotConfig.xml的各部分实例内容如下所示：

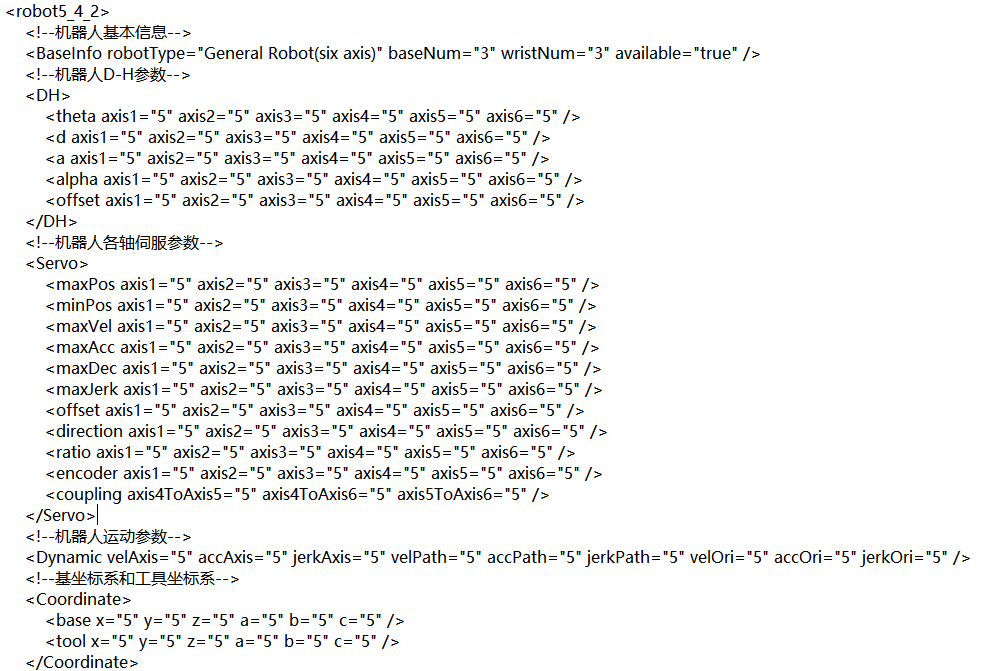


图3.1 配置文件示例图

机器人控制器将对以上配置文件进行解析，获取所加载机器人名称的节点，通过TinyXML类库的相应函数接口进行操作，完成对配置文件对应节点数据的获取。整个参数加载模块的工作流程图如下图所示：

图3.2 参数加载模块工作流程图

由图2.3可看出，机器人控制系统的参数加载模块主要采用TinyXML类库完成了如下操作，将机器人名称、机器人类型、机器人DH参数、机器人各轴极限参数、机器人运动参数以及机器人基座标系相对世界坐标系的位置等参数信息获取之后，赋值到控制器程序的相应变量中，为下一步的逻辑控制操作做准备。

## 3.2 数据传输模块

在控制器实际运行的过程中，需要和示教器进行数据的传输，其中需要与示教器交互的数据如下：

//示教器发送给控制器的数据

int enableState = 0; //机器人使能状态，0：未使能，1：使能

int operateMode = 0; //机器人操作模式，0：手动，1：自动

int runState = 0; //机器人运行状态，0：暂停，1：运行中

int step = 0; //程序执行的方式，0：连续，1：单步

int jog = 0; //手动示教时所选坐标系，0：轴关节坐标系，1：世界坐标系，2：工具手坐标系

int coordinate = 0; //手动示教时选中的坐标，0：未示教，1-6：第x个坐标

int upOrDown = 0; //手动示教时对选中坐标的增大或减小，0：减小，1：增大

int percentage = 20; //相对最大运动参数的百分比

int progLine = 0; //程序执行到的行数

string dataContent = "{}"; //data文件的内容

string progContent = "{}"; //prog文件的内容

//控制器发送给示教器的数据

AXISPOS axisPos = { 0 }; //机器人在轴关节坐标系下的位置坐标

CARTPOS cartPos = { 0 }; //机器人在世界坐标系下的位置坐标

DRIVERPOS driverPos = { 0 }; //机器人在伺服编码器中的位置坐标

DYNAMIC dynamic = { 0 };

int hasReachSingularity = 0; //是否到达机器人的奇异点，0：否，1：是

vector<pair<int, string>> infoReport; //信息报告的内容

这些数据在控制器和示教器之间将通过TCP网络传输协议来进行传输，数据格式在传输过程中遵循peotobuf协议，以提高数据传输的效率。数据传输模块的工作流程图如下图所示：

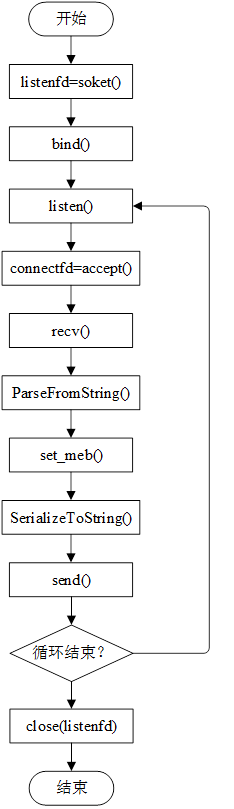


图3.3 数据传输模块工作流程图

如上图所示，控制器作为TCP服务器端，监听来自客户端的请求，当收到客户端的请求时，服务器通过recv()函数，将示教器发送的数据传入控制器的数据缓冲区中，控制器程序从缓冲区中读取数据，进行序列化的操作，将相应的数据赋值给相应的数据变量，到此为止，示教器到控制器的数据传输过程完成。之后将进行控制器到示教器的数据传输过程， 首先，控制器程序通过protoc生成函数接口，将传输的数据反序列，然后通过TCP传输协议的send()函数接口来实现控制器到示教器的数据传输，完成两者相互的数据传输之后，控制器程序将关闭服务器和客户端的进程端口，释放相应的系统资源，结束一轮的数据传输过程。

数据传输模块的技术要点总结如下：

1. TCP服务端的搭建，具体为以下8点：
2. 初始化Windows Socket(WSAStartup)
3. 建立服务端Socket(socket)
4. 与本机进行绑定(bind)
5. 开始监听(listen)
6. 与客户端建立连接(accept)
7. 与客户端进行通信(send,recv)
8. 当通信完成后，关闭连接(closesocket)
9. 释放Winsocket占用的资源(WSACleanup)
10. 数据交互格式采用protobuf协议。Protobuf是一种语言无关、平台无关、可扩展的序列化结构数据的方法，也是一种灵活，高效，自动化机制的结构化数据序列化方法，可类比XML，但比XML更小(3-10倍)、更快(20-100倍)、更为简单。程序中利用的要点如下：
11. protoc:利用协议缓冲区编译器创建一个C++类，方便程序利用C++接口对预定好的数据进行操作。
12. SerializeToString:利用生成的C++函数接口将各个消息体的数据赋值成功之后，利用SerializeToString函数将数据序列化成字符串，然后将字符串传入数据缓冲区，通过TCP服务端所创建好的Socket想客户端发送数据。
13. ParseFromString:由于控制器和示教器之间的数据传输过程是双向的，所以控制器端同样会接受从示教器客户端所传来的字符串数据，这些字符串是在客户端通过步骤2传送过来的，之后通过ParseFromString函数，即可将数据反序列化，从而通过C++类接口的相关函数即可获取各个数据的数值，从而为程序下一步的逻辑操作做准备。

3、ZipPack数据压缩和解压缩：控制器是否解压主要通过客户端传来的标志位transerZip来判断。若为1，控制器将对文件进行解压，过程如下：利用ZipPack库的压缩和解压函数，将客户端通过Socket传过来的数据通过fwrite函数写入压缩文件中，然后通过unPackZipToDir函数将其解压。若为0，控制器将不对文件进行解压。

## 3.3 逻辑控制模块

逻辑控制模块是机器人控制系统中最为重要的一个模块，其主要作用是作为连接上位机（示教器）和RTX64运动控制内核的枢纽，依据用户在上位机上执行的操作，例如：伺服上电、切换坐标系（世界坐标系、关节坐标系、工具坐标系）、调节机器人运动速度、自动/手动模式的切换、程序的运行和停止等等，这些相应的操作信息将通过开发的通讯模块传输到控制器中，控制器依据这些数据信息进行逻辑处理，通过语言解析模块（该模块将在后续章节进行详细阐述），将对上位机用户做运行的程序文本进行文法和语法分析，判断程序编译是否通过，若编译通过，程序将进入到执行阶段，根据用户在上位机所编写的程序，调用内核想用的运动控制API，从而控制机器人的运动。逻辑控制模块的工作流程图如下图3.4所示：



图3.4 逻辑控制模块工作流程图

如上图3.4所示为机器人逻辑控制模块工作流程图，具体设计思路为：

1. 当控制器程序启动时，机器人逻辑控制模块首先会对变量emergeStop进行判断，该变量为程序是否处于急停状态的标志，当其值为1时，代表程序处于急停状态，程序将紧急停止所有轴，并把FIFI缓冲区的未执行的命令全部清空，同时关闭设备，释放所有资源。当emergStop值为0，代表程序处于非急停状态，此时程序将设置机器人的动态倍率参数，同时设置运动参数，上电使能，将IMPC设备打开。
2. 判断activeState,该变量为程序是否处于激活状态的标志，当其值为0时，表示程序处于未激活状态，值得注意的是，未激活状态又分为两种情况，分别是动态未激活状态和静态未激活状态。动态未激活指的是程序在未激活状态之前的状态是激活状态，因此，此时的未激活状态是从激活到未激活的变化过程，因此，需要调用内核指令停止连续插补运动，停止所有轴。静态未激活状态指的是程序在未激活状态之前的状态还是未激活状态，因此，此时的未激活状态不存在变化的过程，从而程序不会对此时的状态做出响应。当其值为1时，表示程序处于激活状态，值得注意的是，激活状态也分为两种情况，分别是动态激活状态和静态激活状态。动态激活状态指的是程序在激活状态之前是未激活状态，存在一个从未激活到激活的变化过程，因此，需要调用内核指令，打开插补缓冲区，设置前瞻参数。静态激活状态指的是程序在激活状态之前还是激活状态，不存在一个状态变化的过程，因此，程序不会对此情况做出响应。
3. 判断startFromActive，该变量代表程序是否从激活到运行的标志，当其值为1时，表示程序从激活状态变为运行状态，此时程序将进入语言解析和运行线程，根据用户在上位机编程的文本语言调用响应的运动控制API。当其值为0时，将继续判断startFromStop的值，该变量代表程序是否从程序停止停止状态变为程序运行状态，当其值为1时，表示程序从停止状态变为程序运行状态，此时将打开连续插补缓冲区，回复电机的运行。当其值为0时，表示程序一直处于停止状态，此时程序将不对该状态做出任何响应。

以上工作流程图和阐述为机器人逻辑控制模块的详细设计思路，可以看出程序在正常运行时，将进入语言解析模块，该模块的阐述将在后续章节详细介绍。

## 3.4 状态读写模块

状态读写模块主要作用是读取和计算电机的相关参数，从而让控制器程序可以将这些相关的参数数值传输到上位机中。状态读写在每个周期的工作流程如下图3.5所示：



图3.5 状态读写模块工作流程图

如上图3.5所示为状态读写模块工作流程图。可看出该模块主要是获取三方面的数据：

1. 轴关节坐标系的位置。首先需要调用内核指令读取编码器的数值，然后根据读取轴的逻辑位置，再根据各轴关节零点相对伺服零点的偏移量计算出轴关节坐标系的位置。
2. 伺服电机的位置。根据机器人在轴关节坐标系下的位置、各轴的运动方向以及减速比计算得出。
3. 工具坐标系相对世界坐标系的位置。该数据主要通过机器人在轴关节坐标系下的位置，调用控制器的正向运动学API计算得出。

## 3.5 语言解析模块

语言解析模块的主要作用是对上位机当前正在运行的程序文件进行词法和语法分析，判断当前所运行程序的文件是否存在单词拼写错误、逻辑错误、词法和文法语法等错误，若存在错误，程序将输出编译不通过的提示，若程序不存在词法、语法的错误，程序将继续进行进行，进入到程序的执行阶段。语言解析模块实际上属于逻辑控制模块的一部分（在阐述逻辑控制模块过程中已进行简单说明），它和逻辑控制模块的关系如下图3.6所示：



图3.6 语言解析模块关系图

如上图所示即为语言解析模块的关系图，用户首先需要设计语义分析树，语义分析树是用户对所定义的数据、变量、符号集合的抽象表达方式，也可以理解为外部程序访问语言解析模块解析之后输出数据的接口。通过这个函数接口对变量、符号等数据进行逻辑存储，访问以及错误判断等。

语法分析树的模型如下图所示：



图 3.7 语义分析树模型

由上图3.7所示为语义分析树的模型，可以看出该分析树是自底向上构造而成的，该模型分析树中每一个节点都是结构体变量，每一个节点的生成都和语言解析模块有关（后续将阐述语言解析模块的设计），通过该分析树可以对程序文本的抽象表达，并且分析程序文本是否存在词法、语法的错误，从而完成编译的过程。

语言分析模块主要有以下两个组件设计而成：

1. Flex词法分析器

该组件的作用是将文本数据分割成一个个的标记token(标识符identifiers,关键字keywords,数字number,中括号brackets,大括号braces,etc)

1. Bison语法分析器

经过Flex语法分析器返回的一个个token,Bison语法分析器将token标记的时候生成语法分析树，因此需要预先定义好所需的语义分析树。

语言解析模块的结构图如下所示：



图3.8 语言解析模块结构图

下面对Flex和Bsion的语法结构进行简要阐述：

1. Flex语法结构

1）definitons:定义段包含了简单名称的声明和开始条件，或者函数头文件

2）rules：包括一组模式和在生成分析器识别相应模式后对相应模式进行处理的C语言动作（Action）

3）user subroutine:用户附加C语言部分，Flex分析器对此不作任何处理，仅仅将至直接拷贝到输出文件的尾部，在这些部分，可定义对模式进行处理的C语言函数、主函数和yylex要调用的函数yywarp()等。如果用户在其他C模块中提供这些函数，用户代码部分可以忽略。

2.Bison语法结构

1）proLogue：可用来定义在动作中使用类型和变量，可以再预处理器命令定义宏，或者使用#include包含干这些事情的头文件。

2）Bison declarations:声明了终结符和非终结符以及操作符的优先级和各种符号语义值得各种类型。

3）Grammer rules:定义了如何从每一个非终结符的部分构建起整体的语法规则。

4）Epilogue:可以包括任何想使用的代码，在ProLogue中声明的函数经常定义在这里，在简单的程序里，剩余的所有程序可以放在这里。

# 关于程序的编译与运行

程序的编译和运行非常简单，主要分为如下几个步骤：

1. 下载Visual Studio2017。
2. 打开RobotController.sln,点击菜单栏中的项目，选择重定向目标解决方案，此时Visual Studio会自动识别程序所在主机的Windows SDK版本，点击确定即可，如下图所示为示例：

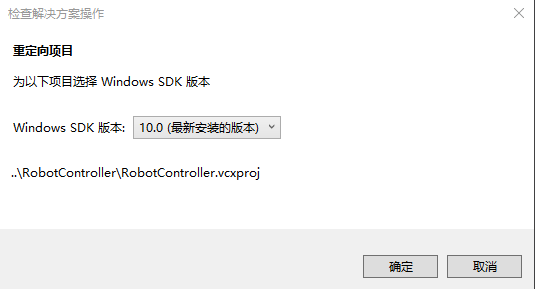


图4.1 重定向目标解决方案

1. 点击菜单栏中的生成按钮，选择清理解决方案选项，清除掉之前可能存留的解决方案，再选择重新生成解决方案选项，此时，VS会根据程序相应的依赖项和外部库进行链接，生成可执行程序，此时编译阶段已完成。
2. 点击菜单栏中的调试按钮，选择开始调试或者开始执行选项，即可成功运行程序，控制器程序即进入等待状态，等待上位机的操作，作出响应。