摘　要

基于单处理器的开放式系统已经成为机器人控制系统的发展趋势，系统由专用机器人语言和专用微处理器的封闭式结构走向通用、实时、多任务、模块化、具有人机接口和网络功能的开放式结构。

随着微处理器技术的发展，PC 机以其标准的接口、合理的硬件结构及优异的性能价格比，成为机器人控制系统开发的首选硬件平台。对于基于 PC 的机器人控制系统，系统的软件构架尤其重要，不仅要有友好的人机界面，而且必须是实时多任务的和体系开放的。本文提出基于 Windows NT+RTX 平台的机器人伺服系统的开发方法。

本文首先进行了机器人运动分析及仿真，以确定机器人手臂的有效工作区域和验证各种运动的可行性，并采用 Visual C++6.0 进行开发，可以实现仿真和机器人控制的有效结合。

接着论述了双臂教学 SCARA 机器人伺服系统的设计方法，并进行了机器人的伺服控制研究。系统硬件主要由 PC 机、硬件控制箱和机器人本体组成，采用“PC + 控制卡”的控制平台，控制卡有 I/O 卡、D/A 卡和数据采集卡。系统软件采用 Windows NT+RTX 实时平台，系统的所有功能都在这个平台下实现。其中，软件系统由人机界面、Win32 通信层、RTSS 通信层、运动控制层组成。人机界面和 Win32 通信层在 Win32 非实时子系统内实现，RTSS 通信层和运动控制层在 RTSS 实时子系统内实现，依靠共享内存等进程间通信机制来保证非实时系统和实时系统的通信。伺服控制部分在运动控制层完成，主要论述了伺服系统的建模及控制算法，给出了相应的运行结果，并分析了系统的实时性能。系统开发结果表明：Windows NT+RTX 平台完全能够满足控制系统对实时性的要求，开发基于此平台的机器人伺服系统是完全可行的。

关键词：机器人控制系统　开放性　RTX　机器人仿真　目标轨迹规划法

ABSTRACT

Open architecture control based on Single processor has been becoming the essential requirement for modern robotic control system. Old architecture which use proprietary language and CPU has changed into open architecture with many traits such as popularity, real time, multitask, modularization, human-machine interface, network, etc.

With the development of processor, PC becomes the first choice of the control kernel of robot, for its normative configuration, reasonable structure and excellent performance-price ratio. For robot controller based on PC, the framework of software is the most important. It has friendly human-machine interface, and is real time, multitask and open architecture. The paper introduces a develop method of a Two-Arm SCARA Manipulator servo system based on Windows NT+RTX.

In order to determine work area of the manipulator and guarantee feasibility of the motion of it, the paper firstly goes on with analysis of kinematics on it and carries through computer simulation. Visual C++6.0 is used to develop the system, with an aim of realizing the combine of the simulation and the manipulator motion according to it.

Secondly the servo system architecture of the manipulator is discussed. Hardware of the system consists of PC, hardware control box and the manipulator. "PC+control cards" based platform is adopted, and control cards consists of I/O card, D/A card and data acquisition card. Windows NT+RTX platform is used to develop the system, and all the function of the system is implemented on the platform. Software of the system consists of human-machine interface, communication layer in Win32, communication layer in RTSS and motion control layer. Human-machine interface and communication layer in Win32 are realized in Win32 sub system; Communication layer in RTSS and motion control layer are realized in RTSS sub system which is real time. Share memory method is brought forward to solve the communication problem between the two sub systems. Servo control is implemented in motion control layer, modeling of servo system and control algorithms are discussed, the corresponding results are given and the performance of real time is analyzed. The implement of the system indicates that Windows NT+RTX platform is able to full fill real time of servo system, and the development of servo system of the robot is feasible.

Key words: control system of robot, openness, RTX, computer simulation of robot, planning method of target track.

学位论文独创性声明：

本人所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果。与我一同工作的同事对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。如不实，本人负全部责任。

论文作者（签名）：赵小英　　2006 年 3 月 21 日

学位论文使用授权说明

河海大学、中国科学技术信息研究所、国家图书馆、中国学术期刊（光盘版）电子杂志社有权保留本人所送交学位论文的复印件或电子文档，可以采用影印、缩印或其他复制手段保存论文。本人电子文档的内容和纸质论文的内容相一致。除在保密期内的保密论文外，允许论文被查阅和借阅。论文全部或部分内容的公布(包括刊登)授权河海大学研究生院办理。

论文作者（签名）：赵小英　　2006 年 3 月 21 日

第一章 绪论

1.1 课题背景

机器人技术是伴随着工业自动化的浪潮而兴起的一门技术，它集成了机械学、电子学、仿生学、人工智能、计算机学、自动控制工程等学科，发展成为一门新的学科——机器人学。

工业机器人已经是一种高科技的成熟产品，50 年代末由美国研制发明，60 年代成为产品；70 年代日本快速发展，产、销量都成为第一大国。中国工业机器人的发展有将近 20 年的历史，从第七个五年计划（1986——1990）开始，我国政府将工业机器人的发展列入其中，并且为此项目投入大量的资金。在第七个五年计划中，我国研究开发并且制造了一系列工业机器人，有北京机械自动化研究所设计制造的喷涂机器人，广州机床研究所和北京机床研究所合作设计制造的点焊机器人，大连机床研究所设计制造的氩弧焊机器人，沈阳工业大学设计制造的装卸载机器人等。这些机器人的控制器，都是由中国科学院沈阳自动化研究所（SIA）和北京科技大学机器人研究所开发的。同时一系列的机器人关键部件也被开发出来，如机器人专用轴承，减震齿轮，直流伺服电机，编码器，DC_PWM 等。与此同时，由航空工业部支持的用于汽车制造的机器人项目的发展也非常迅速，一系列汽车生产用的喷涂机器人，点焊机器人，氩弧焊机器人等，相继由哈尔滨工业大学和 Fenghua 机床制造公司设计制造，并成功投入使用。 除此之外，另一批由当地政府支持的机器人研发队伍也在大力发展机器人技术。最新统计数据显示，目前我国有将近 160 个组织在做机器人的研究。大力展开机器人的研究，将为我国机器人产业化输入源源不断的动力[1]。

1.2 国内外研究现状

1.2.1 教学机器人

伴随着汽车制造业、计算机行业、重机械制造业和航空航天业的发展，机器人技术日新月异。机器人技术与仿生学结合起来，出现了能够迅速对周围障碍物做出反应的机器蝎子、可以进行太空探索或排除地雷的机械蟑螂；机器人技术和纳米技术结合起来，出现了医学上可以治疗各种疾病的纳米机器人[2]。沈阳新松机器人自动化股份有限公司研发的 SIASUN-RD120 机器人具有 6 个自由度，最大额定工作负荷 120kg，水平作业半径 2.6 米，重复定位精度±0.4 毫米，属于一种适用于点焊及搬运、码垛等作业的大负荷

垂直多关节通用工业机器人产品，如图 1-1。它具有结构合理、可靠性高、性能先进、实用性强的特点。首批制造的 6 台 RD120-A 点焊机器人，已用于一汽集团红旗轿车焊装线，从而使我国自主开发的点焊机器人达到新水平。RD120-A 点焊机器人每分钟点焊打点能力为 40~50 点/分钟，达到了国际先进水平。

图 1-1 SIASUN-RD120 点焊机器人

国内工业机器人的研究已经取得可喜成绩，不过教育机器人的研究仍处于起步阶段。率先进行机器人教育的有香港城市大学，他们采用"趣味"机器人——微型电子鼠和乒乓球机器人进行教学。微型电子鼠主要由传感器、电机、处理器、软件组成，每个部分有多种选择。电子鼠的各个部分的任意组合几乎是无限的，因此可以实施电子、机械和控制软件应用的综合教育[3]。上海交通大学机器人研究所在这方面也做的比较好，他们自主研发了 EDUROBOT 680 II 五自由度教学机器人和二自由度 SCARA 教学机器人，功能多样，适合教学不断深入的需要。让人兴奋不已的是，各种足球机器人比赛已在国内各高校展开并取得可喜的成绩。此外，国内在科普方面做的比较好的公司有固高科技（深圳）有限公司，他们设计的一套 GRB-200 两自由度教学机器人，其机械结构合理，并且利用该公司生产的 GT-400 运动控制卡能够较好的实现各种基本的控制功能，总的来说比较适合于教学和实验。在国外，教学机器人的研究已经受到很大的重视，因为一个国家要大力发展机器人技术的话，首先就应该从教育起步。日本是世界上机器人教育水平和机器人文化普及水平最高的国家之一。培养机器人的兴趣是从幼年开始培训，而且是教育干部系统必须履行的职责之一。从高中阶段开始深入学习机器人方面的知识，并一直持续到大学。不仅每所大学具有高水平的机器人研究和教学内容，且每年举行多种不同档次的机器人设计和制作大赛，通过大赛培养了大批机器人技术研究和应用人才，使日本的机器人技术走到了世界前列。日本牧野洋设计的 SCARA 机器人已经在教育和工业界普及并形成产业化[4,5]。

1.2.2 机器人控制系统的开放性

随着计算机控制技术、软件开发技术和先进制造技术的发展，人们逐渐认识到机器人控制系统专用性带来的弊病，迫切需求具有软硬件配置灵活、功能扩展简便、基于统一规范的系统，即向通用性、可扩充性发展的开放式系统。传统上的工业机器人所采用

的控制系统基本上是设计者基于自己的独立结构而开发的，它采用了专用的计算机、专用的机器人语言、专用的操作系统，这种封闭式结构限制了它的可扩展性和灵活性，不便于对系统进行扩展和改进[6]。美国于1987年推出了下一代控制器NGC(Next Generation Controller）计划，首次提出了开放式体系结构的概念。随后为了构建开放式体系结构，欧共体国家的 22 家控制器开发商、控制系统集成商和科研机构于 1992 年联合提出了OSACA（Open System Architecture for Control with Automation System）计划；美国克莱斯勒、福特和通用三大汽车公司于 1994 年开始了一项名为“开放式、模块化体系结构控制器（OMAC）”的计划；日本东芝机器公司、丰田机器公司和 Mazak 三家制造商以及日本 IBM、三菱电子以及 SML 信息公司共同提出了开放系统计划（OSEC）[7,8,9]。

IEEE 对开放式控制系统的定义是：开放式控制系统提供这样一些功能，它们能在不同厂商的各种不同平台上运行，能与其它系统相互兼容，并且具有一致风格的用户交互界面。根据这个定义，一般认为开放式控制系统应该具备以下特点[7,9]：

◆ 互操作性(Interoperability)

不同功能模块提供标准化接口、通信和交互机制，能以标准的应用程序接口运行于系统平台之上，并具有平等的相互操作能力，可以协调工作。

◆ 可移植性(Portability)

系统软件所具备的功能与设备无关，即应用统一的交互模型、数据格式、控制机制，构成系统的各功能模块可来源于不同的开发商，并且通过一致的设备接口，系统各功能模块能运行于不同供应商提供的硬件平台之上。

◆ 可伸缩性(Scalability)

系统的功能、构成可以灵活设置和修改，用户既可以精简其结构以适应低端应用，也可以增加软件模块或者硬件构成功能更强的系统。

◆ 互换性(Interchangeability)

系统的各硬件、软件模块的选用不是唯一的，可根据其功能、可靠性及性能要求从不同的供应商采购，且对整个系统的正常运行毫不影响。

当前，机器人开放式系统硬件平台大致可以分为两类：基于 VME 总线的系统和基于 PC 总线的系统。采用通用 PC 作为机器人控制器的主控计算机的优点是：成本低，具有开放性，完备的软件开发环境和丰富的软件资源，良好的通讯功能，用户基础广泛。近几年，日本、美国和欧洲一些国家都在开发具有开放式结构的机器人控制器，如日本安川公司基于 PC 开发的具有开放式结构、网络功能的机器人控制器，MOTOMAN、SEIKO 等大公司均把基于 PC 的机器人作为主要发展方向[10]。我国 863 计划智能机器人主题也已对这方面研究立项。

1.2.3 单处理器体系结构的机器人控制器

当前在机器人控制系统方面，主要有以下几种实现途径：

第一种是使用 PLC ，它被添加到主要平台上，而且所有获取数据和计算采用单片机完成。在国外广为应用的一个系统是 dspace 系统，它的目的是为实验室而用，也用于原型的实现[11]。PLC 的使用减少了研究人员和工程师的编程量。其主平台对来自终端用户或者其它平台如服务器的通信，采取自由管理。这种平台的缺点是高成本和低柔性。考虑到操作系统的能力，PLC 中的自适应控制参数不能自动更新，只能被主机更新。此外，当前商业可行的可编程序控制器（PLCs）控制系统价格比较高。

第二种类型是基于微机的多处理器控制系统。目前，国内外存在的基于 PC 的工业机器人及教学机器人控制系统几乎全部采用多处理器（CPU）结构，其中又以两级主从处理器结构形式居多。如 Intelledexj 机器人控制系统采用 Intel8086/8087 微处理器，每一关节有两个 8080 微处理器作伺服控制。它所用的语言为类似 BASIC 的 ROBOTBASIC，具有坐标变换、速度设定、直线和圆弧插补等功能。美国 GRACO 机器人公司的 OM500 机器人控制器采用三个 8086CPU 和二个 8085CPU。我国精密 1 号装配机器人控制器，采用 Intel 公司的 iSBC386/12 系列计算机，并用上下两级分布式计算机结构(Multibus 总线) [12]。固高公司的 GRB-200 两自由度教学机器人是由一台 PC 机及插在 PC 总线上的 GT-400 运动控制卡组成，运动控制卡上包含专用的数字信号处理器（DSP）。上述机器人控制器都采用了多 CPU 结构，并且能够达到很好的控制性能。但是，他们还存在着以下的不足之处：1）控制系统结构比较复杂，成本较高，可靠性也有所影响；2）控制系统下层 CPU 开放性较差，不利于功能扩展和底层控制的实验教学。

第三种类型是基于单处理器结构的微机通用实时系统。机器人控制器发展基于计算机发展，又落后于计算机发展。当今 Pentium 及 PentiumProCPU 的运算速度和处理数据能力是 8086CPU 的数百倍，用单一的 CPU 替代早期多 CPU 结构已具可能。对于单处理器体系结构控制器的研究，国内研究得虽然不多，国外却有一些大中型控制软件公司致力于基于 PC 的单处理器控制器的实现。其中具有典型代表的是美国 Entivity 软件公司的 Visual Logic Controller（VLC）软件，提供了一种模块化的控制器设计方法，而用户只需通过搭建流程图的方法就能完成整个控制系统的构建。但是，该软件的机器人控制器模块不是专门为机器人设计的，难以进行功能的进一步扩充，不适合推广到教学当中。不过当前流行的操作系统 Windows NT、UNIX 和 Linux 都扩展有实时系统，可以完成实时任务。采用通用操作系统来完成机器人实时任务，具有成本低、开发周期短、人机界面友好、系统扩充灵活等优点。

第四种类型是基于单处理器结构的微机专用实时操作系统。它使用微机上的处理器和时钟，完成数据的记录和计算校正系统的发送信号。出于商业和工业的目的，出现的专用实时操作系统有 QNX 软件系统中的 QNX™，lynx 实时系统中的 Lynxos，和 WindRiver 系统中的 VxWork™[13]。这些实时操作系统的优点是它们可升级的运行软件、高精度的计时器和从制造商那里得到的好的支持。这些系统的缺点是它们的较高的开发成本和较长的开发周期。

综合以上考虑，本文研究的机器人伺服系统采用第三种类型的结构，采用基于

Windows NT+RTX 实时平台来实现系统的各种控制任务。同时我们可以看出，以往机器人伺服系统大多是建立在传统封闭式结构的基础上，基于开放式结构的机器人伺服系统的研究目前正处于起步阶段，进行机器人伺服系统的研究是很有意义的。

1.3 论文研究目的和主要内容

本课题主要探讨和解决基于 Windows NT+RTX 单处理器实时环境下的机器人开放式控制系统的建模与实现。目的在于开发出一套高效可靠的能满足当前工业控制需要的全软实现的控制系统。论文分为以下几个部分：

第一章主要介绍教学机器人技术在国内外的研究现状，讲述基于单处理器体系结构的机器人控制器的国内外现状，并说明本论文的研究内容、目的和意义。

第二章介绍机器人的结构，并进行运动学正解和逆解，然后在运动学正解和逆解的基础上运用 Visual C++6.0 完成运动区域仿真和各种运动轨迹仿真。

第三章说明机器人单处理器结构控制器的思想、优点，进行了机器人控制系统硬件结构的介绍和实时系统的选择，说明其能够满足运动控制系统实时性的原理和机制。

第四章介绍 Windows NT 和 RTX 操作系统各自的基本特性、体系结构以及通信和中断的处理机制，着重分析 Windows NT 在实时性方面的局限性及 RTX 优越的实时性能。

第五章阐述基于 Windows NT+RTX 单处理器实时环境下的机器人伺服系统的软件设计。

第六章探讨伺服系统的具体控制，给出控制算法和相应的运行结果。然后对系统进行性能分析，以保证控制系统的可靠性、稳定性和精确性。

第七章结论与展望

1.4 课题的研究意义

基于 Windows NT+RTX 的系统是一种可靠、高效的实时操作系统，它为开发出一套高效、稳定、可靠且能满足工业控制需要的全软实现的控制系统提供了可能，论文探讨了这一工业控制的新的实现方法和方向，具有一定的实用价值和研究价值。本文通过对教学机器人伺服系统的建模及实现来研究伺服系统的相关特性，以便于更好的对伺服系统进行控制，不断提高伺服系统的性能和对工业环境的适应能力。因此，研究本课题具有一定的理论指导意义。

本课题在单处理器体系结构下解决实时控制、多任务调度及高速通信的问题，来适应教学和科普机器人开放性好、成本低的需要；再者，解决我国当前受教育群体多和教育经费有限这一矛盾所致的教学设备落后的问题。

第二章　双臂 SCARA 机器人运动分析及仿真

本章将进行双臂教学 SCARA 机器人机械本体介绍、手臂运动分析、工作空间分析，并利用计算机进行工作空间的确定和工作空间内手臂的各种运动仿真。之所以通过机器人运动分析、工作空间分析来展开全文的内容，是因为这一步骤是机器人系统设计的重要基础，它关系到机器人各种运动是否能顺利的实现，也有益于机器人机构的进一步改进。

特别要指出的是，机器人同其他机械产品一样，机构的运动不是任意的，我们必须考虑其运动干涉约束，这就是设计机器人系统的另一个重要因素——安全性，这在本章也将给予讨论。为了使工作空间的分析更有效、直观，本章借助计算机工具进行了仿真。在计算机仿真这部分内容里，本文将提出一套机器人机构的简化方案，完成机器人各种手臂姿态规划，分析机器人运动存在的运动干涉约束，最后运用运动学正解和逆解进行软件仿真设计及实现，得出手臂的具体工作区域，并在工作区域内进行各种运动仿真。

2.1　机器人机械本体介绍

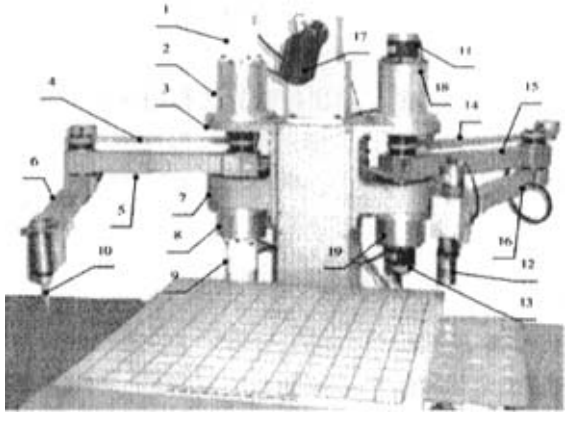


图 2-1　双臂教学机器人

本文研究的机器人是上海交通大学机器人所研制的多功能双臂教学 SCARA 机器人，左右臂分别是一 SCARA 机械手，如图 2-1 所示。它由以下部件组成：左小臂 16、左大臂 15、右小臂 6、右大臂 5、左臂平行四边形外杆 14、右臂平行四边形外杆 4、左臂步进电机 11 和 13、右臂伺服电机 1 和 9、左臂电机谐波减速器 18 和 19、右臂电机谐波减速器 2 和 8、左臂末端棋抓 12，右臂末端绘图笔 10、肩 3 和 7、摄像机 17。

2.1.1 传动机构

与传统 SCARA 机器人相比[14]，机器人的运动就是两只机械手大小臂的摆动。在机构设计上，左臂和右臂用螺钉固定在肩的两侧，摄像机安放在肩的中间，肩安放在基座上。左右臂的机械结构相同，都是由电机、小臂、大臂、小臂电机减速器组、大臂电机减速器组和平行四边形机构组成。不同的是左臂使用步进电机 11、13 驱动，采用开环控制方式；右臂使用交流伺服电机 1、9 驱动，采用闭环控制方式。下面通过右臂传动来具体说明：伺服电机 1 由电机谐波减速器组 2 减速后通过平行四边形机构带动右小臂 6，平行四边形机构由外杆 4、右大臂 5 及两者间的连杆组成[15]；伺服电机 2 由电机谐波减速器组 8 减速后直接带动右大臂 5。

机器人机构的设计具有如下的特点：

（1）由于左右臂采用不同类型的电机，可以进行对比研究、性能比较，引导学生进行实际的操纵控制、程序编程，对教学特别具有重要的意义。

（2）由于左、右臂的控制可以同时进行，它们可以协调运动，绘图笔可以进行绘图和写字等教学内容，同时左臂还可以进行人机对弈，不过这一功能的实现有待进一步研究。

（3）小臂采用平行四边形机构进行驱动，使得大小臂的关节控制可以独立进行，大大方便了控制的有效实施，并有利于实际性能的改善。

（4）由于大臂电机减速器组和小臂电机减速器组设计在同一轴线上，因此大臂电机减速器组和小臂电机减速器组都可以安装在肩上，而不需象传统设计那样把小臂电机减速器组放在大臂上，从而降低了大臂的负载。

2.1.2 末端执行机构

末端执行机构有左臂末端棋抓 12，右臂末端绘图笔 10，如图 2-1。

左臂末端棋抓主要完成棋抓下落、棋抓上升、吸附棋子和放开棋子四个动作。其中，棋抓下落和棋抓上升是由手臂末端的小型步进电机直接驱动的，吸附棋子和放开棋子是由电磁铁来操控的。取棋的过程是这样的：手臂移动到取棋点，手臂末端的小型步进电机动作，将棋抓下移到棋子上方约 5mm 处，接着电磁铁动作，吸附棋子，小型步进电机再次动作，将棋抓上移。放棋子的过程和取棋子相似，唯一不同的是，电磁铁动作的时候不是吸附棋子而是放开棋子。

右臂末端绘图笔主要完成写字绘图功能，绘图笔装置仅完成两个动作：绘图笔落下和绘图笔抬起，通过电磁铁来控制。当电磁铁电源接通时完成“落笔”过程，电磁铁电源断开时完成“抬笔”过程。

2.2 机器人运动分析

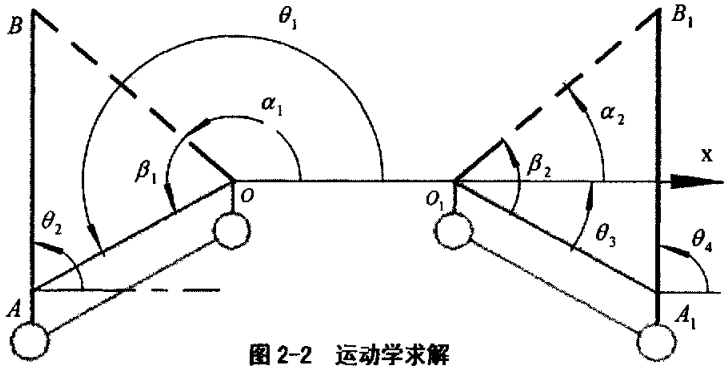


图 2-2 运动学求解

由于工作区域仿真运用运动学正解进行，首先要进行运动学正解，即要由手臂运动角度来确定手臂末端位置。如图 2-2 所示，AB 是左小臂，OA 是左大臂，A_1B_1 是右小臂，O_1A_1 是右大臂，采用笛卡儿坐标系，O 为原点，x 轴水平向右，y 轴垂直 x 轴向上。设 $OA=l_1$，$AB=l_2$，$O_1A_1=l_3$，$A_1B_1=l_4$，已知 l_1、l_2、l_3、l_4、O_1 点坐标(x_{O1},y_{O1})及变量 θ_1、θ_2、θ_3 和 θ_4，由图 2-2 可知，左臂运动学正解即求点 B 坐标(x_B,y_B)：

$$\begin{bmatrix} x_B \\ y_B \end{bmatrix} = \begin{bmatrix} \cos\theta_1 & \cos\theta_2 \\ \sin\theta_1 & \sin\theta_2 \end{bmatrix} \begin{bmatrix} l_1 \\ l_2 \end{bmatrix} \tag{2-1}$$

同理可以求出右臂运动学正解即求点 B_1 坐标(x_{B1},y_{B1})：

$$\begin{bmatrix} x_{B1} \\ y_{B1} \end{bmatrix} = \begin{bmatrix} x_{O1} \\ y_{O1} \end{bmatrix} + \begin{bmatrix} \cos\theta_3 & \cos\theta_4 \\ \sin\theta_3 & \sin\theta_4 \end{bmatrix} \begin{bmatrix} l_3 \\ l_4 \end{bmatrix} \tag{2-2}$$

为了验证机器人工作区域和传动角的范围，有必要进行机器人运动学逆解。逆解主要应用在手臂某种运动轨迹的实现中，即需要从轨迹点反解出手臂关节要运动的角度[16]。

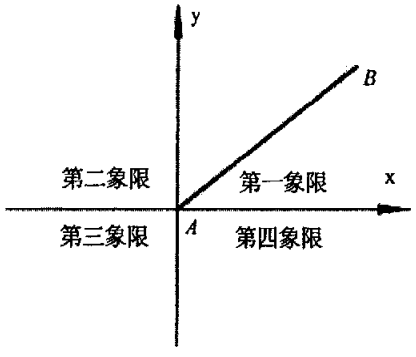


图 2-3 手臂象限的约定

由于手臂在不同姿态时，运动学逆解是不同的，我们首先引入手臂象限的约定。如图 2-3 所示，每条手臂的象限约定是以其转动轴心为原点，x 轴和 y 轴分别平行于图 2-2

中的 x 轴和 y 轴，图中手臂 AB 的转动轴心是 A 点，在第一象限。

首先我们参照图 2-2 讨论下手臂可能在的象限，图 2-2 是手臂的回零状态，即手臂处于外限位置，只能向中间收缩，不能再向外围张开。左小臂由于不能继续向外张开，只能处于 y 正半轴上、第一象限、x 正半轴上和第四象限；左大臂可以处于第三象限、x 负半轴上、第二象限、y 正半轴上和第一象限；右小臂允许置于 y 正半轴上、第二象限、x 负半轴上和第三象限；右大臂可以活动在第四象限、x 正半轴上、第一象限、y 正半轴上和第二象限。

分情况讨论左右臂的运动学逆解时，会发现有多种组合，与其一一列举，不如用计算机式的语言来实现。以下就从方便编程的角度来讨论逆解。

参考图 2-2，设 $OB=l_0$，OA、AB 同上所设，已知点 B 坐标(x_B,y_B)、l_1 和 l_2，求左臂运动学逆解 θ_1 和 θ_2，由图 2-2 可知：

$$\theta_1=\beta_1+\alpha_1 \tag{2-3}$$

其中，$l_0=\sqrt{x_B^2+y_B^2}$，$\beta_1=\cos^{-1}\dfrac{l_0^2+l_1^2-l_2^2}{2l_0l_1}$

$$\begin{cases}\alpha_1=\tan^{-1}\dfrac{y_B}{x_B}, & \text{当点B在第一象限，}x_B>0;\\ \alpha_1=\dfrac{\pi}{2}, & \text{当点B在y轴上，}\quad x_B=0;\\ \alpha_1=\pi+\tan^{-1}\dfrac{y_B}{x_B}, & \text{当点B在第二象限，}x_B<0;\end{cases}$$

则可求出 A 点坐标：

$$\begin{cases}x_A=l_1\cos\theta_1\\ y_A=l_1\sin\theta_1\end{cases} \tag{2-4}$$

设 $k_1=\dfrac{y_B-y_A}{x_B-x_A}$，$\theta_2$ 可以根据 A，B 两点坐标来求：

$$\begin{cases}\theta_2=\tan^{-1}k_1, & \text{当手臂AB在第一四象限},x_B>x_A;\\ \theta_2=\dfrac{\pi}{2}, & \text{当手臂AB与Y轴平行，}\quad x_B=x_A;\\ \theta_2=\pi+\tan^{-1}k_1, & \text{当手臂AB在第一四象限},x_B<x_A;\end{cases} \tag{2-5}$$

右臂运动学逆解同样可以根据图 2-2 求出，设 $O_1B_1=l_{01}$，O_1A_1、A_1B_1 同上所设，已知点 B_1 坐标(x_{B1},y_{B1})、l_3 和 l_4，求左臂运动学逆解 θ_3 和 θ_4，由图 2 -2 可知：

$$\theta_3=\beta_2+\alpha_2 \tag{2-6}$$

其中，$l_{01}=\sqrt{x_{B1}^2+y_{B1}^2}$，$\beta_2=\cos^{-1}\dfrac{l_{01}^2+l_3^2-l_4^2}{2l_{01}l_3}$

$$\begin{cases} \alpha_2=\tan^{-1}\dfrac{y_{B1}}{x_{B1}}, & \text{当点B在第一象限，} x_{B1}>0; \\ \alpha_2=\dfrac{\pi}{2}, & \text{当点B在y轴上，} \quad x_{B1}=0; \\ \alpha_2=\pi+\tan^{-1}\dfrac{y_{B1}}{x_{B1}}, & \text{当点B在第二象限，} x_{B1}<0; \end{cases}$$

则可求出 A_1 点坐标：

$$\begin{cases} x_{A1}=l_3\cos\theta_3 \\ y_{A1}=l_3\sin\theta_3 \end{cases} \tag{2-7}$$

设 $k_2=\dfrac{y_{B1}-y_{A1}}{x_{B1}-x_{A1}}$，$\theta_4$ 可以根据 A_1，B_1 两点坐标来求：

$$\begin{cases} \theta_4=\tan^{-1}k_2, & \text{当手臂}A_1B_1\text{在第一四象限,} x_{B1}>x_{A1}; \\ \theta_4=\dfrac{\pi}{2}, & \text{当手臂}A_1B_1\text{与Y轴平行,} \quad x_{B1}=x_{A1}; \\ \theta_4=\pi+\tan^{-1}k_2, & \text{当手臂}A_1B_1\text{在第一四象限,} x_{B1}<x_{A1}; \end{cases} \tag{2-8}$$

2.3 机器人运动仿真

在机器人控制系统设计中，机器人的运动分析及仿真是确定机器人工作区域及在工作区域内进行各种运动解析的准备工作，是非常重要的环节。目前机器人仿真一般采用 Envision 等专用仿真软件，很难取得仿真中的数据，而且它只局限于某些功能的实现，难以进行功能的进一步扩充，不适合推广到教学当中；也有用 OpenGL 进行三维实体仿真[17]，但由于编程复杂，需要利用 Cortona SDK 等三维工具箱辅助造型，虽然真实感强，但往往会造成执行速度慢，可靠性、可移植性差等问题。为了在仿真的过程中能够获取仿真数据，同时避免三维仿真存在的问题，加之本文中的机器人运动是左右两臂的平面运动，机器人仿真采用二维仿真，应用 Visual C++6.0 进行程序设计。由于机器人的底层控制软件也用 VC 开发，因此可将其和仿真软件相结合，实现对机器人运动的控制。

2.3.1 机器人机构的简化

为了实现仿真智能化——手臂仿真运动过程中自动识别极限位置，有必要使机器人机体、手臂和四连杆机构简化，以便获取相关位置参数。由于机器人的运动就是手臂做的平面运动，其上各部件可以简化成平面图形，简化后如图 2-4 所示。

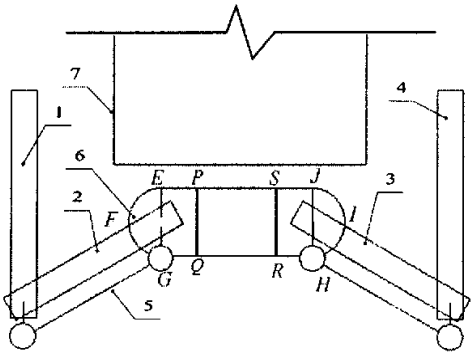
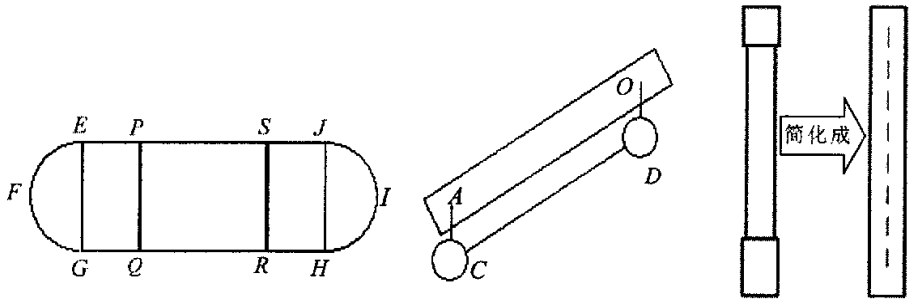


图 2-4　机器人简化图



（a）　　（b）　　（c）

图 2-5　机体的简化

a. 机体的简化　如图 2-4 中 *EFGHIJ* 球场形区域，具体如图 2-5(a)，*PQRS* 矩形区域高于机体 6 的其它部分。

b. 四连杆机构的简化　左臂四连杆的简化机构由图 2-4 中大臂 2、外杆 5、两个关节圆、关节圆与大臂 2 间的连杆组成，具体如图 2-5(b)。考虑到四连杆中除大臂 2 外其它杆件在仿真智能化中不起作用，故用直线代替；由于四连杆关节参数在仿真中需要，以圆的形式给出。

c. 手臂的简化　如图 2-4 中手臂 1、2、3、4 所示。由于仿真过程中需要的手臂参数主要是手臂外四角的参数，所以将手臂由两头大中间小的部件转化成一矩形部件，具体如图 2-5(c)，矩形部件的宽度是手臂宽度的最大值。

2.3.2　手臂姿态规划

手臂姿态规划的主要任务是构造不同姿态的手臂。在利用运动学正解求出机器人手臂的关节点后，便可进行手臂姿态规划，即由手臂的关节点求简化手臂所在矩形的四角坐标。手臂的姿态不同，四角坐标的求法也不同，对处于第一象限的手臂姿态，如图 2-6 所示，已知轴线 *AB* 到 *M'N'* 边的距离 d_0、关节点 *A* 到 *MM'* 边的距离 d_1、关节点 *B* 到 *NN'* 边的距离 d_2 和关节点坐标 $A(x_A,y_A)$、$B(x_B,y_B)$，依据几何关系，求得手臂四角坐标 $N(x_N,y_N)$、$M(x_M,y_M)$、$M'(x_{M'},y_{M'})$、$N'(x_{N'},y_{N'})$为：

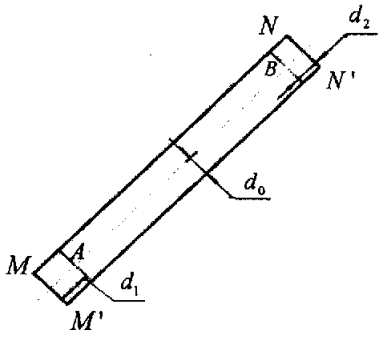


图 2-6 手臂姿态规划

$$\begin{bmatrix} x_M \\ x_{M'} \\ x_{N'} \\ x_N \end{bmatrix} = \begin{bmatrix} 1 & 0 & -d_1 & -d_0 \\ 1 & 0 & -d_1 & d_0 \\ 0 & 1 & d_2 & d_0 \\ 0 & 1 & d_2 & -d_0 \end{bmatrix} \begin{bmatrix} x_A \\ x_B \\ s_2 \\ s_1 \end{bmatrix} \tag{2-9}$$

$$\begin{bmatrix} y_M \\ y_{M'} \\ y_{N'} \\ y_N \end{bmatrix} = \begin{bmatrix} k & & & \\ & k & & \\ & & k & \\ & & & k \end{bmatrix} \begin{bmatrix} x_M \\ x_{M'} \\ x_{N'} \\ x_N \end{bmatrix} + \begin{bmatrix} y_A - kx_A + d_0 s_0 \\ y_A - kx_A - d_0 s_0 \\ y_A - kx_A - d_0 s_0 \\ y_A - kx_A + d_0 s_0 \end{bmatrix} \tag{2-10}$$

式中，$s_1 = \dfrac{k}{s_0}, s_2 = \dfrac{1}{s_0}, k = \dfrac{y_B - y_A}{x_B - x_A}, s_0 = \sqrt{k^2 + 1}$

当图 2-6 中手臂旋转到第二象限时（图上标注保持不变），求得手臂四角坐标矩阵为：

$$\begin{bmatrix} x_M \\ x_{M'} \\ x_{N'} \\ x_N \end{bmatrix} = \begin{bmatrix} 1 & 0 & -d_1 & d_0 \\ 1 & 0 & -d_1 & -d_0 \\ 0 & 1 & d_2 & -d_0 \\ 0 & 1 & d_2 & d_0 \end{bmatrix} \begin{bmatrix} x_A \\ x_B \\ s_2 \\ s_1 \end{bmatrix} \tag{2-11}$$

$$\begin{bmatrix} y_M \\ y_{M'} \\ y_{N'} \\ y_N \end{bmatrix} = \begin{bmatrix} k & & & \\ & k & & \\ & & k & \\ & & & k \end{bmatrix} \begin{bmatrix} x_M \\ x_{M'} \\ x_{N'} \\ x_N \end{bmatrix} + \begin{bmatrix} y_A - kx_A - d_0 s_0 \\ y_A - kx_A + d_0 s_0 \\ y_A - kx_A + d_0 s_0 \\ y_A - kx_A - d_0 s_0 \end{bmatrix} \tag{2-12}$$

继续旋转图 2-6 中手臂到第三象限，手臂四角坐标矩阵为：

$$\begin{bmatrix} x_M \\ x_{M'} \\ x_{N'} \\ x_N \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_1 & d_0 \\ 1 & 0 & d_1 & -d_0 \\ 0 & 1 & -d_2 & -d_0 \\ 0 & 1 & -d_2 & d_0 \end{bmatrix} \begin{bmatrix} x_A \\ x_B \\ s_2 \\ s_1 \end{bmatrix} \tag{2-13}$$

纵坐标矩阵同式(2-12)。

图 2-6 中手臂旋转到第四象限，手臂四角坐标矩阵又变为：

$$\begin{bmatrix} x_M \\ x_{M'} \\ x_{N'} \\ x_N \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_1 & -d_0 \\ 1 & 0 & d_1 & d_0 \\ 0 & 1 & -d_2 & d_0 \\ 0 & 1 & -d_2 & -d_0 \end{bmatrix} \begin{bmatrix} x_A \\ x_B \\ s_2 \\ s_1 \end{bmatrix} \tag{2-14}$$

纵坐标矩阵同式(2-10)。

2.3.3 存在的运动干涉约束

2.3.3.1 空间约束

在解决手臂仿真智能化问题时，参考图 2-4 主要考虑以下几点：

(a) 各手臂做外扩运动即远离工作台 7 时，四连杆机构关节圆是否碰到机体的 *PQ* 部分和 *SR* 部分；

(b) 小臂 1、4 做内缩运动即靠近工作台 7 时，其内侧是否碰到机体圆弧面 *EFG* 或者 *HIJ*；

(c) 小臂 1、4 做外扩运动时，四连杆机构外杆是否碰到大臂转动轴；

(d) 小臂 1、4 做内缩运动时，其末端是否会碰到 *EJ* 部分；

(e) 大臂 2、3 做内缩运动时，其内侧是否会碰到 *P* 点、Q 点所在的机体侧棱。

通过图 2-2，来讨论左小臂是如何实现运动干涉约束的，即如何在程序循环中识别极限位置并跳出：

if($\theta_1<\theta_0$&& $x_b > x_a$&& $y_{N'}<r+e_0$) break;　if($\theta_1>\theta_0$&& $x_b > x_a$&&$d_3< r+e_0$) break;

代码中，θ_0 为图 2-4 中左小臂 1 内侧刚好与机体内壁 *EPSJ* 重合时的 θ_1 值，r 为图 2-4 中机体 *EFG* 弧的半径，e_0 为机体与手臂为避免碰撞而允许的最小间隙，d_3 为原点到左小臂内侧的距离。

第一条 if 语句考虑左大臂处于第一象限、y 正半轴上和第二象限位置时左小臂的运动约束，即小臂末端不能碰到机体；第二条 if 语句考虑左大臂处于第三象限时左小臂的运动约束。

2.3.3.2 传动效率约束

由于公式(1) 、(2)中的θ_1和θ_2受到四连杆机构中传动角的约束，考虑到机构传动效率，有必要进行传动角的分析。由图2-7所示左臂姿态，*OP*是机器人本体，小臂*AB*运动而大臂*OA*固定不动，小臂*AB*速度方向*V*沿*C*点垂直于杆*CA*(小臂*AB*和*CA*杆固连在一起且共线)。*OACD*是四连杆机构，*F*是*CD*杆对*CA*杆的力，它可以分解成法向力F_n和切向力F_t。根据定义，在不计运动副摩擦和构件质量的情况下，机构从动件受力方向*F*和受力点速度方向*V*所夹的锐角α，称为机构在此位置的压力角(Pressure Angle)。压力角越小，机构的传力性能越好，效率越高。在连杆机构中，为了度量方便，常用传动角γ(Transmission Angle)来衡量机构的传力性能。传动角γ定义为压力角α的余角。大多数机构在运动过程中，传动角是变化的。由图2-7可得传动角γ：

$$\gamma = \delta = \theta_1 - \theta_2 \qquad (2\text{-}15)$$

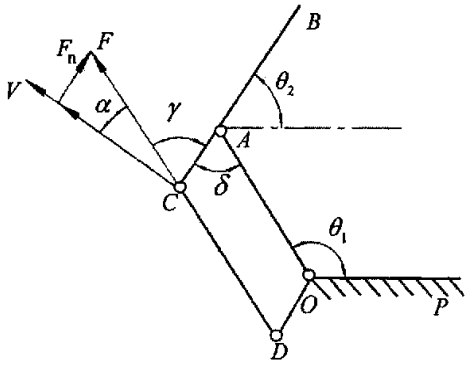
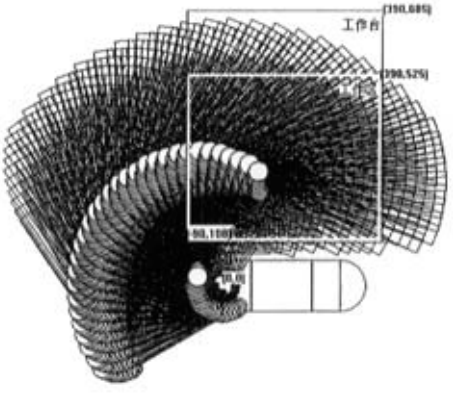


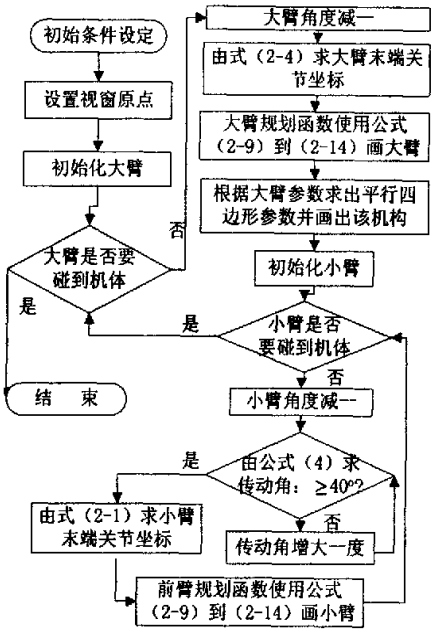
图 2-7　传动角的求解

同样可以求出左右臂其它姿态时四连杆机构的传动角；可以发现，任何情况下机构的传动角总等于同侧大小臂的夹角。为保证四连杆机构具有良好的传动性能，规定其最小传动角 γ=40°[18]。

2.3.4　工作区域仿真的实现



(a)



(b)

图 2-8　左臂工作区域仿真

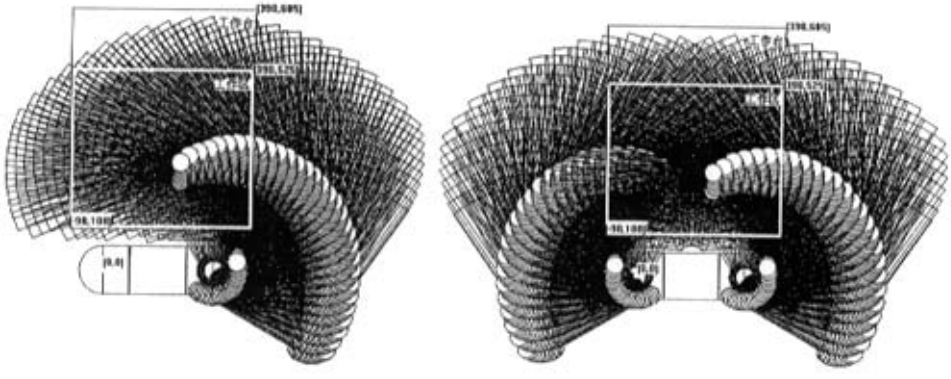
为了使仿真具有可扩充性、方便教学和学生编程，采用普遍应用的 Visual C++6.0 软件平台。通过左右臂仿真来确定手臂在工作台上实际可工作的区域，区域仿真采用运动学正解的方法求出手臂关节点坐标，通过手臂姿态规划函数得出手臂坐标参数，然后画出相应姿态的手臂。当然，手臂不能任意的摆动，我们必须依据空间约束规则判定手

臂运动的具体范围，不让手臂和机体发生碰撞。

左臂工作区域仿真结果见图 2-8(a)，手臂工作区域是白线框所围矩形区域，外围黑线框是工作台所处区域。从图上可以看出，在仿真过程中，手臂并没有与机体发生“碰撞”，说明仿真有效。左臂仿真框图见图 2-8(b)，框图中菱形部分使用来进行手臂运动空间约束。

实际上，左臂形成的工作区应该是左臂区域仿真过程中在工作台上留下的阴影部分，为了比较方便识别，用白线框的矩形区域来代替。以下工作区域的确定也是用工作台上阴影区域的最小矩形面积来代替。

右臂的工作区域的仿真框图的结构类似于左臂，其仿真结果见图 2-9(a)，手臂工作区域也是白线框所围矩形区域。同图 2-8(a)一样，在仿真过程中，手臂并没有与机体发生“碰撞”，说明仿真有效。图 2-9(b)是左右臂先后进行区域仿真后形成的工作区域仿真图。从图上可以看出，左右臂的工作区域仿真形成的区域关于工作台对称，且左右臂各自仿真的工作区和左右臂共同仿真形成的工作区相同。



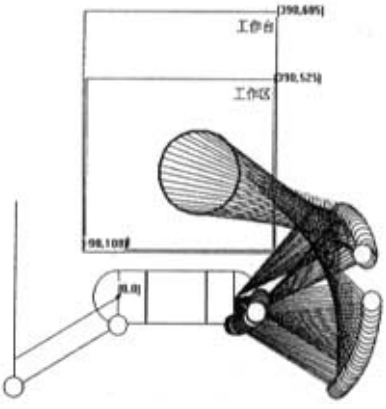
(a)　　(b)

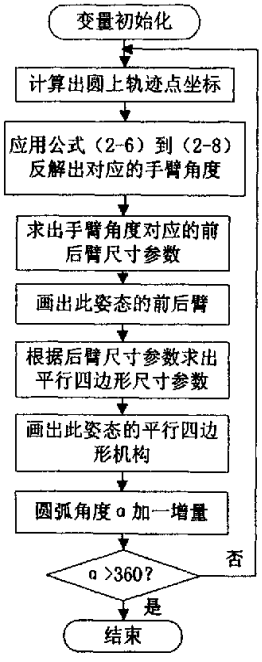
图 2-9　右臂工作区域和左右臂工作区域仿真

2.3.5　工作区域内各种运动仿真的实现

图 2-10(a)是运用运动学逆解在工作区域内进行的圆弧运动仿真；图 2-10(b)是圆弧运动仿真程序设计框图。仿真的原理是基于微直线段逼近曲线的方法，直线段越短，越逼近曲线；圆弧角度的增量越小，大小臂的摆角增量越小，微直线段越短，圆就越圆。

由于机器人的底层控制软件也是用 VC 平台开发，故可以和仿真相结合来控制机器人手臂的运动；圆弧运动中传动角数据运用 MATLAB 工具进行分析，画出仿真中传动角的变化曲线如图 2-11，由图可见，传动角在 43°到 85°之间变化，保证了机构的传动效率。





（a）　　（b）

图 2-10　右臂圆弧运动仿真

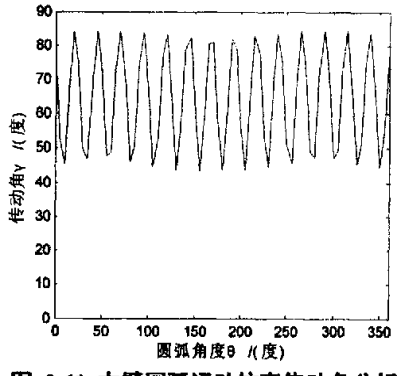


图 2-11 右臂圆弧运动仿真传动角分析

此外，运用运动学反解还进行了圆弧插补运动仿真、直线插补运动仿真、直线运动仿真、示教运动仿真，分别见图 2-12(a)、(b)、(c)、(d)。

仿真软件实际运行性能良好，由于采用通用 VC 平台进行设计，且不采用辅助仿真软件，有效地解决了可靠性、可移植性差等问题，并能以文件的方式灵活地获取仿真中的数据。

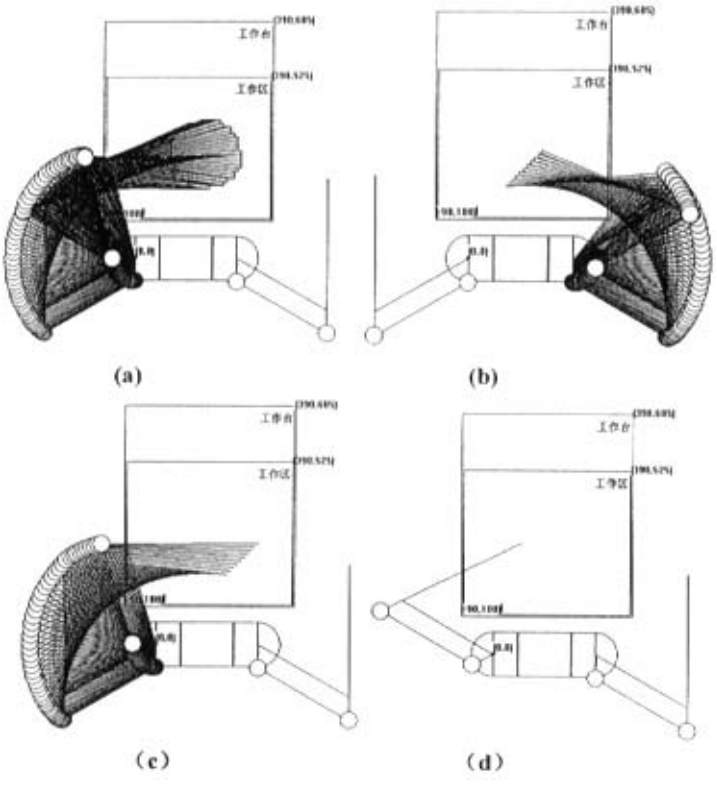


图 2-12　左臂直线运动仿真

2.4　本章小结

本章主要围绕三部分内容进行展开。首先，对于双臂 SCARA 机器人的机械本体作了大致的介绍，包括双臂结构、平行四边形结构以及电机的同轴设计等结构特色。紧接着，进行了双臂教学 SCARA 机器人手臂运动学分析和综合。第三部分建立了机器人的运动模型，给出了机器人各种手臂姿态的求解矩阵，并利用通用软件平台 Visual C++ 6.0 进行了机器人手臂工作区域的运动仿真和在工作区域内的各种运动的仿真；并可以把机器人仿真和实际控制结合起来，丰富了机器人教学。

第三章　控制系统的总体结构

机器人机电系统是由许多类型的驱动器和不同类型的传感器组成的，驱动器类型有电动、气动、液压和压电电机等，传感器类型有数字脉冲编码器、电位计、转速计、加速度计、力传感器和张力测量装置等。机器人机电系统要求从所有的传感器读取数据，求得基于传感器数据的控制信号，然后发送这些控制信号到相应的驱动器。传感器数据采样和计算速度必须非常快和达到高的精度，这样才能保证系统的稳定性和可靠性。因此，有必要在现有硬件结构的基础上选择合适的软件结构以使系统的各种控制顺利实现。

必须重点指出的是，机器人的控制系统的稳定性、实时性和可靠性是必须考虑的。控制系统的好坏不仅直接关系到机器人任务能否有效实施，还与机器人工作时的安全性紧密相关。

3.1　机器人控制系统硬件结构

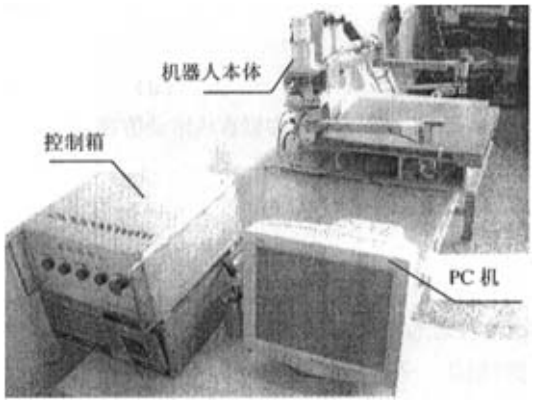


图 3-1 机器人硬件总体框架

机器人的硬件总共分三个组成部分：总控制器即 PC 机、控制箱和机器人本体，如图 3-1 所示。

PC 机是机器人控制的核心，主要来实现单处理器结构的思想，即基于 PC 机的控制系统。主要完成以下功能：

（1） 通过数据采集卡，读取伺服电机的反馈脉冲，转化为电机的位置信号和速度信号，并通过相关运算，转化为右臂末端的位置和速度信号。

（2） 经由 I/O 控制卡，读取左右手臂的极限位置，以便手臂准确回零和提供限位保护。此外，步进电机控制脉冲和伺服电机位置控制模式下控制脉冲的发送也是通过 I/O 卡，左右臂末端执行器的动作也是通过 I/O 卡进行控制。

（3） 使用摄像头获取下棋图像。这主要应用在下棋的过程中，通过获取相邻两步的图

像信息，来分析棋局的变化。

（4） 当伺服驱动器工作在速度控制模式下时，运用 DA 控制卡来给定伺服电机的控制电压。

控制箱是机器人驱动信号的直接来源，控制箱的硬件主要包括两大部分：伺服电机驱动器及其附属电路、步进电机驱动器及其附属电路。前者主要完成伺服电机智能驱动和保护，后者主要完成步进电机的驱动及保护。伺服驱动器使用时还要考虑各种参数的调节，使它达到较理想的性能。

机器人本体是机器人的机械部分，主要是接收各种驱动信号，通过机构传动部件来驱动机器人手臂进行各种运动，主要介绍见上章 2.1 节。

3.2 基于单处理器控制结构的实时系统选择

3.2.1 实时系统要素

由于机器人运动对实时性的要求比较高，机器人控制系统必须是一个实时系统。实时操作系统是一个在不可预测的外部事件达到后，在可预测的时间内及时响应的操作系统。对于一个实时性要求高的系统，不管它是要完成周期性的任务，还是要完成非周期性的任务，都要求有较强的定时约束，要求在一定时间内必须完成对事件的处理，如果有延迟，可能会造成严重后果。为了满足实时特性，系统必须有一些基本的要素[19]:

（1） 满足实时响应

一个事件按照中断方式到来时，系统能在特定的时间期限内做出响应，并且其实时内核必须提供最坏情况下的时间响应的保障即性能优化。

（2） 多任务

由于伺服系统的事件的异步性，能够运行许多并发任务是很重要的。在进行计算机控制的过程中，多个任务对应着多个线程的运行。系统内核按一定的算法分配CPU给这些任务来获得良好的运行效果，并满足相关任务的时间期限。

（3） 抢占调度

抢占调度是在多任务的基础上提出来的，多任务有其固有的执行顺序，我们按照这个顺序给各任务分配一优先级，各任务执行时按照基于优先级的抢占调度算法来进行调度，有效地解决了多任务协作运行的问题。

（4） 高效的任务间的通信与同步

为了保证多任务的协作运行和共享资源，必须提供各任务间有效的通信与同步机制，使任务间共享的资源不被非法更改。

3.2.2 开放式控制器的思想

第一章给出了开放式控制系统的概念，这节将从控制器的层次来讨论开放性，以便选择开放式实时系统和给出基于开放式控制器的软件结构。开放式结构机器人控制器是指: 控制器设计的各个层次对用户开放，用户可以方便的扩展和改进其性能。 其主要思想有[20]:

（1） 利用基于非封闭式计算机平台的开发系统，如Sun,SGI,PC。有效利用标准计算机平台的软、硬件资源为控制器扩展创造条件。

（2） 利用通用的操作系统和控制语言。采用标准操作系统和控制语言，从而可以改变各种专用机器人语言并存且互不兼容的局面。

（3）采用标准总线结构, 使得为扩展控制器性能而必须的硬件, 如各种传感器、I/O板、运动控制板，可以很容易的集成到原系统。

（4） 利用网络通讯，实现资源共享或远程通讯。

根据以上思想和绪论中谈到的开放式控制系统的特点我们可以设计具有开放式结构的机器人控制器。模块化是现代控制系统设计和构建的重要方法，通过模块化设计，我们可以把系统的各种功能划分成若干模块，各模块在设计上可以相互独立，在功能上可以相互联系，既可以缩短开发周期，又可以大大减少开发成本。此外，模块化大大方便了对系统的后期维护、扩充和更改。系统开放式控制器的模块化设计将在第五章详细论述。

3.2.3 实时系统的选择

实时系统的选择是基于单处理器体系结构控制系统开发的关键环节，它直接关系到机器人控制系统的开发难度、系统性能以及应用的广度。在绪论1.2.3节已经讨论过，基于PC的实时系统有专用实时系统和通用操作系统加实时扩展，由于专用实时系统开发成本比较高、开发周期比较长和人机界面不友好等缺点，我们设计时选择通用操作系统加实时扩展构成的系统。

目前，运行在PC机上的通用操作系统主要有DOS、Windows NT、UNIX和LINUX，由于DOS操作系统是单任务操作系统，而机器人控制系统不可避免的出现底层运动控制和人机界面同时运行的情况，即要求操作系统是多任务操作系统，操作系统只在Windows NT、UNIX和LINUX中选择。。

◆ Windows NT 操作系统

Windows NT 代表了个人计算机操作系统的新潮流，支持多种硬件平台、多个 CPU、支持 32 位和 64 位的多任务，其基于优先级抢占式多任务调度方式和功能强大的可视化软件开发工具为软件开发提供了很大便利；Windows NT 很重视面向对象设计（OOD）的概念，这种方法很容易在进程间共享数据和资源，并且对于未授权的访问提供对资源

的保护；Windows NT 通过采用虚拟地址空间的方式在稳定性方面有了较大提高，每个应用程序可以使用 2GB 的内存；由于 Windows NT 系统本身不是实时操作系统，许多商家纷纷推出 NT 的实时展系统，如 VenturCom 的 RTX，Tenasys 的 INtime 等。其中，以 VenturCom 的 RTX 应用的比较广泛。

◆ UNIX 操作系统

UNIX 系统是一个多用户的分时操作系统，可移植性好、可靠性强、具有可装卸的树型分层结构文件系统。UNIX 系统将所有外部设备都当作文件看待，分别赋予它们对应的文件名，用户可以像使用文件那样使用任一设备，而不必了解该设备的内部特性；UNIX 开始设计时考虑了同时提供多人使用的要求，为此 UNIX 系统中设置了许可权。UNIX 在优先级调度、存储器管理、时钟与定时器、同步与互斥和 I/O 控制等方面的实现机制，使得 UNIX 在实时应用方面受到不同程度的限制，Digital UNIX 通过附加一个功能强大的实时应用库，扩充了 UNIX 实时功能[21]。

◆ LINUX 操作系统

LINUX 是一套类似于 UNIX 的、源代码开放的、免费的、支持多种硬件平台、多个 CPU、支持 32 位和 64 位的多任务、多用户的网络操作系统。但是，LINUX 本身并不是一个实时的操作系统，它的核心的不可抢占性、采用基于固定时间片的可变优先级调度算法、支持虚拟内存、10ms 的任务调度时间精度以及在使用临界资源时关中断时间比较长的特性在很大程度上限制了其在实时控制领域的应用。目前已经有 RTLINUX、KURT、RTAI 等多种实时 LINUX 操作系统[22]。尽管如此，这种系统并不是十分支持不同的设备接口。由于该系统上的资源十分有限，应用范围并不广。

通用操作系统具有良好的可移植性并可以运行在多种硬件平台之上，以上操作系统都提供了强大的网络功能和良好的用户界面，拥有丰富的软件资源和软件开发工具，其开发能力远远优于专用的实时操作系统。本伺服系统的设计选用应用最广泛的 Windows NT 4.0 操作系统及其 RTX 实时子系统作为 PC 机的操作系统。

3.3 本开放式控制系统的总体结构

机器人开放式控制系统总体结构见图 3-2，人机控制界面处于 Windows NT 系统内，机器人的具体控制处于 RTX 实时系统内，这两层构成机器人的软件控制系统，软件控制系统通过硬件抽象层和硬件打交道，即通过驱动各种控制卡来操作机器人和获取机器人的信息，大致说明见本章 3.1 节。需要指出的是，本文伺服系统的设计是针对有伺服电机的右臂，左臂暂不考虑。因此，不用到摄像头和左臂驱动。

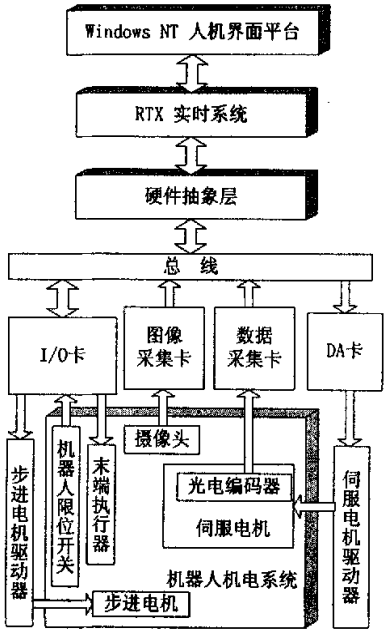


图 3-2 机器人控制系统总体结构

3.4 本章小结

本章首先介绍了机器人控制系统硬件结构，讲述了各硬件组成部分的功能和相互关系；接着从系统的实时性和开放性出发，综合比较当前实时化的各种通用操作系统，选择Windows NT+RTX作为本机器人伺服系统的软件平台；最后给出了系统的总体结构框图，为以后的伺服系统的软件设计打下基础。

第四章 Windows NT 及 RTX 实时子系统

4.1 Windows NT 操作系统

Windows NT 操作系统的主要设计者是 David Cutler，在此之前他是 DEC VMS 操作系统的主要设计师，NT 的许多原理都体现了他设计 VMS 的经验和他改进 VMS 的想法。

Windows NT 的最早版本于 1993 年 7 月问世，它是一种全新的操作系统。这种操作系统在当时真正实现 32 位 PC，从根本上提高了计算机的运行能力。在此之前，大多数 32 位 PC 只能使用 16 位的操作系统和 16 位的软件。

1995 年，微软发布了 Microsoft Windows NT 3.51 版本，该版本修正了前一版中的一些 bug，并且增加了一些新的功能包括文件和目录的压缩，还增加了对硬件的支持。版本 3.51 是 Windows NT 销售市场占有率上的转折点，标志着它开始向当时享有的市场份额迅速攀升。

Windows NT4.0 是微软公司在 1996 年 7 月在 NT3.51 基础上推出的基于网络的操作系统，一经问世，就以其配置方便、安全稳定以及友好的界面赢得了市场的认可[23]。

4.1.1 Windows NT 体系结构

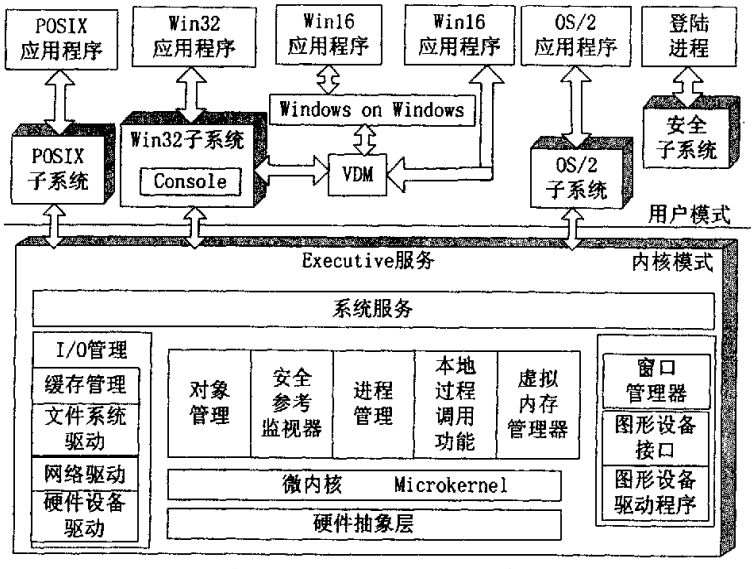


图 4-1 Windows NT 总体结构

Windows NT 的各个组成部分采用了 Carnegie Mellon 大学 1985 年开始研制的叫做 MACH 的项目结构，也就是采用了微内核结构，如图 4-1 所示。

Windows NT 采用的是模块化的结构，也就是说这个结构是通过把独立的、截然不同的组件集合起来而形成的。组件的独立性使得开发者不用重写或是重新编译整个系统，就可以把操作系统从一种处理器的平台移植到另一种处理器的平台上。Windows NT 系统的运作有两种模式：内核模式和用户模式。

内核模式是系统运行中的高优先级模式，如图 4-1 中长直线段下面的部分。运行于该模式的组件能够直接访问系统中的各种硬件和内存，包括所有运行于用户模式下的进程的地址空间。内核模式中的很大一部分就是图 4-1 中的 Windows NT Executive，它包括硬件抽象层、微内核和 Executive 服务如系统服务、I/O 管理、对象管理、进程管理、虚拟内存管理和窗口管理等。

用户模式是系统运行中的低优先级模式，如图 4-1 中长直线段上面的部分。运行于该模式的组件不能够直接访问硬件，只能访问自己的地址空间。即使是访问自己的地址空间，也要通过内核模式来申请访问。在用户模式下的程序需要访问系统资源时，它使用操作系统的 API（应用程序接口）来申请访问系统资源，然后等待 API 完成（或者拒绝）它的申请。如图 4-1 所示，Windows NT 下的各种子系统如 Win32 子系统、POSIX 子系统、OS/2 子系统和安全子系统。

4.1.2 WINDOWS NT 操作系统的主要功能和特点

Windows NT 现在已经成为 Microsoft 公司的主要操作系统，Windows 系列软件的升级版也都是基于 Windows NT 的结构，并且越来越获得广泛的应用。Windows NT 是一个支持多个 CPU 多种硬件平台、32 位单用户、多任务操作系统。当前基于 Windows 的 Office 软件、数据库软件已经获得广泛的应用，其编程软件平台 Visual C++已经成为各级院校、各大公司及编程爱好者乐意采纳的平台。总的来说，它有以下特点[24]：

（1）应用范围广

可以运行在单用户、单处理器的 PC 机上，也可以运行在多用户、多处理器的网络环境上。

（2）Windows 图形用户界面(GUI)

它的界面是用户熟悉的图形用户界面。

（3）支持多种硬件平台

Windows NT 支持上千种基于 x86 芯片的单处理机系统、多处理机系统、MIPS、PowerPC、DEC Alpha 系统等。NT 对外围设备的支持也比较广泛，包括视频卡、网卡、SCSI 卡、通讯、多媒体、磁带设备、打印机等。

（4）广泛的兼容性

Windows NT 和 DOS、POSIX、OS/2、16 位 Windows 及 32 位 Windows 保持良好的兼容性，这些功能的实现是通过引入子系统的概念，如图 4-1。

（5）先进的客户机/服务器模型

Windows NT 是微软公司的拳头操作系统，它率先采用了一种计算模型：应用的处理和执行采用内部共享的方式，并通过网络提高运行效率和灵活性，显然 NT 是一种内部客户机/服务器模型，在 NT 中，内部过程协同负责操作系统的运行。

（6）动态连接共享库

Windows NT 对可执行代码采用了动态连接方式，多个可执行代码可以共用一份可共享的程序库文件里的程序库代码。使用这种连接方式可以让可执行文件占用更少的硬盘空间，尤其是对那些调用许多库程序的大型程序而言，更能发挥硬盘及系统的效率。

4.1.3 WINDOWS NT 与其他操作系统的主要区别

Windows NT 许多原理上体现了 VMS 系统的思想，而 VMS 系统在设计上又体现了 UNIX 的许多优秀思想。不可否认，Windows NT 和 UNIX 在某些方面是有共同点的。由于 LINUX 系统是类 UNIX 系统，我们就 Windows NT 和 UNIX 系统进行下比较。

在结构上，Windows NT 是模块化和可扩展的操作系统，虽然 UNIX 系统也是模块化的，但是在系统集成和管理方面存在一些缺陷，而 NT 用如下方法克服了 UNIX 系统存在的缺陷[26]：

（1）紧集成

各种子系统的功能都是平缓而有效地集合在一起的。

（2）各个部分是独立的一块

内存中保存的始终是立即需要的代码，这样不仅提高运行速度，而且提高了物理内存的使用效率。

在应用上，Windows NT 更加强调图形用户界面和桌面应用程序的开发。

4.1.4 WINDOWS NT 在实时性方面的局限性

基本的 Windows NT 操作系统可以定制到满足几乎任何用途，通过硬件的限制和应用混和，NT 系统可以驱动一个频繁访问的大型数据库，也可以驱动来自网络连接、键盘、鼠标上的异步事件，但不可能按硬实时响应。即使这样，NT4 仍可以迅速地处理事件，满足一般的时间响应。

Windows NT 设计初衷是强调系统整体高性能，因此采用了中断、多任务、子系统、堆内存管理、虚拟存储管理、内存映射文件、DMA、客户机/服务器模型、结构化异常处理等机制。但 NT 也正因此丧生了系统行为的确定性和可预测性，NT 普遍被认为不适合开发实时应用。

对硬实时问题来讲，Windows NT 系统存在如下几个缺陷[27]：

（1）中断

中断的优先级要高于任务执行的优先级，故由于中断发生具有不确定性，导致被中

断任务的执行时间也变得不确定了。这种不确定性对于具有时间约束的实时系统可能是致命的。Windows NT 提供了一种延迟过程调用 DPC（Deferred Procedure Call）机制，允许驱动程序在实际中断返回后完成它们的处理，对通用操作系统这显然是可行的，而这对于实时系统却是致命的缺陷。

（2）定时器

Windwos NT计时器的时间间隔被用来更新时钟并且每一个间隔作为一个固定的时间段加在时钟上面，且保持时间间隔增量的精度在0.1us。这将导致一个积累误差，并且造成相对于日时钟的小度漂移。由于这个漂移，Windows NT系统周期性的读取COMS上的日时钟时间并做一个漂移更正，但是这对于实时系统来说是不允许的。

（2）处理机调度

Windows NT 采用基于优先级的调度算法，其核心是一组优先级可调整的队列。调度算法会逐渐调整各个队列的优先级，使得低优先级队列中的任务最终也能被执行。在这种调度算法中，由于无法准确地预测一个任务需要等待多长时间才能被执行，以及多长时间能够执行完毕，因此这种调度算法是无法满足对实时任务调度的。

（3）虚拟内存管理

Windows NT 使用了基于页面的虚拟存储器技术。在虚存机制中，当页面失效后，必然伴随换页操作，这给任务的实时性能带来了负面影响。页面失效的处理过程是非常复杂的，涉及众多因素，如失效中断、磁盘操作、系统存储容量等。而频繁的页面失效会导致无法准确预测一个任务的执行时间。

（4）直接内存存取(DMA)

DMA 技术大大提高了系统的整体吞吐率，有效减少处理器对 I/O 设备操作的干预，提高了处理器利用率。但由于 DMA 必须访问主存储器，因此在 DMA 传输期间，I/O 设备和 CPU 是分时争用主存储器的。DMA 传输方式为准确预测 CPU 完成一个任务的时间增加了不确定因素，无法达到实时系统的要求。

4.2 RTX 实时子系统

RTX是VenturCom公司针对Windows NT的弱实时性而开发出来的基于NT的实时子系统，它通过加入"硬"实时能力而替Windows NT填补这一漏洞。通过扩展Windows NT的操作系统，RTX使得应用软件的构件或者模块具有确定的和高速的响应次数，并且同其他非实时应用程序构件一起工作在Windows NT系统下。Ventur Com公司从Windows NT硬件抽象层开发出RTX硬件抽象层，使得RTX可以直接对硬件进行操作，同时在此硬件扩展层上形成实时子系统RTSS（Real Time Sub System）。实时子系统（RTSS）在概念上和其他Windows NT子系统（如Win32、POSIX、WOW、和DOS）相似，它有自己的执行环境和API函数。不过RTSS在一个重要领域里与Windows NT不同：它使用自己的

实时线程调度程序，而不是Windows NT的线程调度程序。

4.2.1 RTX 的特点

RTX 具有如下特点[28]：

（1）在RTSS环境中，RTSS线程是程序执行的基本单位。Win32线程不能控制RTSS线程，因为RTSS线程的句柄在RTSS环境下才有效。

（2）RTX设备I/O端口的编程接口允许由用户进程来控制数据的流动而不需要转交到内核程序完成。这种特性消除了为每一个设备都创建驱动程序的必要。I/O端口提供了轮流直接与外部设备交互的方法。

（3）RTX物理内存映射接口把一块物理内存映射到应用程序虚拟地址空间中。这将使得应用程序直接访问一块物理内存，就好像在应用程序中有一块内存一样。而Windows NT支持基于没有独立I/O空间的处理器，更确切地说，它的端口被映射入内存地址。

（4）实时进程间通信机制（IPC）提供了最灵活的选择，因为Win32和RTSS进程可以通过同样的接口访问一个实时的驱动器，甚至可以同时访问。

（5）RTX内存分配程序总是分配锁定的内存，来消除对分配页误操作的可能性。

（6）RTX时钟和计时器设施提供精确的计时服务，使线程实时完成与外部时钟相关的周期性任务。RTX时钟被设计成保持一种与某种计时器周期同步的速率（也就是，此计时器已知在什么时候停止）。这就可以计算计时器终止值和计时器滞后时间，消除了Windows NT系统时钟相对于日时钟小度漂移的弊端。

（7）实时进程可以直接响应外部的中断，这种能力允许一个设备被一个用户进程完全控制，并消除写一个核心级驱动程序的必要。

4.2.2 RTX 的实现机理

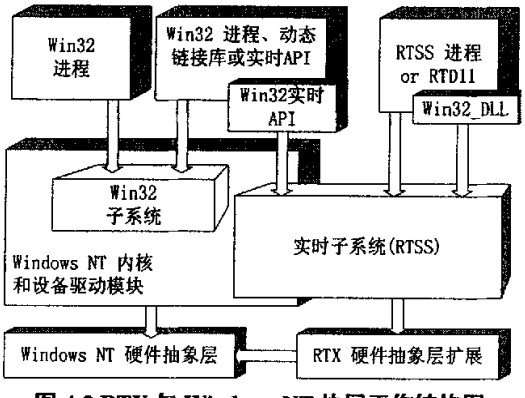


图 4-2 RTX 与 Windows NT 协同工作结构图

图 4-2 描述了 RTX 与 Windows NT 协同工作的结构关系。在硬件层，RTX 通过扩展 Windows NT 的硬件抽象层来直接操作硬件，从而获得硬实时能力；在软件层，RTX 通过与 Windows NT 下的 Win32 子系统通信而完成同 Windows NT 应用程序通信。从图上可以看出，RTSS 建立在 RTX 硬件抽象层扩展的基础上，而独立于 Windows NT 系统。Windows NT 的 Win32 子系统通过调用实时 API 函数来与实时子系统交互；RTX 实时系统可以通过发布 Win32 动态链接库来将实时子系统嵌入到 Windows NT 系统中，也可以通过 RTSS 进程与 Win32 进程的 IPC 机制来与 NT 交互，而这种传送方式是最快的。

4.2.3 RTX 的应用程序编程接口（API）

（1）包括 Win32 和实时编程接口(RTAPI)

RTX 支持一部分 Win32 编程接口（API）函数，加上它提供了一套实时编程接口函数，称作 RTAPI；RTX 编程接口基于 Win32 编程接口，它允许开发人员用他们的 Win32 经验；Win32 和 RTSS 进程对 RTX 全部支持。大多数应用程序包括至少两个进程一起工作：一个是基于 Win32 的进程（利用它图形界面的优点和仅用 Win32 函数的进程），一个是基于 RTSS 的来完成实时处理任务的进程。

（2）提供了三种可执行工程:RTSS 应用程序、RTSS DLLs 和 RTDLLs

RTDLL 是一个实时子系统应用程序，它用来产生一个输出库，实时子系统和其他应用程序可以调用该输出库中的函数，它可以通过 LoadLibrary 和 FreeLibrary 动态加载与卸载 。RTSS DLL 是一个 RTX 进程，可以在系统启动时调用或者在 Windows NT 下的一个 C/C++程序里调用。

（3）支持各种实时库

RTX 支持各种实时库，提供一个基于 VC 的 C 实时库。RTSS 进程可以静态链接这些库，前提是这些库不包含不支持的 Win32 函数。

（4）支持 Unicode 编码

RTX 支持 Unicode 码的应用程序。RTSS 进程可以使用 wmain()函数而且可以接收宽字节的参数输入。

（5）支持单处理器和多处理器 Windows 系统。

RTX 产品的内核不仅支持单处理器系统，而且在 Win32 环境的普通平台下能实现硬实时性能。RTX 支持多处理器的实时版本，不仅包括单处理器版本的所有特性，而且开发了 Intel MPScompliant 多处理器系统，从而使 Windows NT 和 RTX 环境都得到了改善。

4.2.4 RTX 和 WINDOWS NT 间的通信机制

如图 4-3 所示，RTX 通过 Windows NT 的 Win32 子系统和其交互。Win32 进程和 RTX 进程通过共享内存机制来交换数据，通过内存分配和内存锁定来保护交换的数据。在使

用共享内存的同时 Win32 进程和 RTX 进程需要进行进程间通信，要用到计时器和时钟服务及同步对象。计时器和时钟服务包括时钟服务、计时器服务和休眠服务；同步对象有信号量、事件对象和互斥对象。

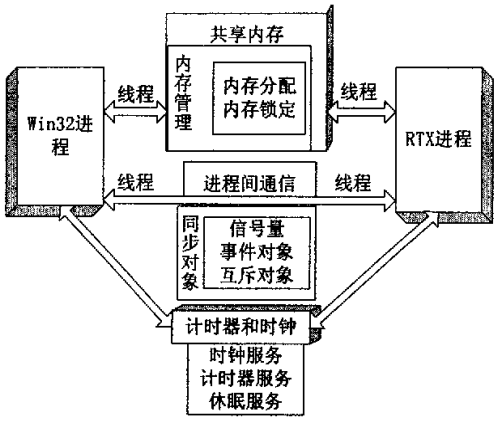


图 4-3 RTX 和 WINDOWS NT 间的通信机制

RTX 采用如下具体机制与 Windows NT 进行通信：

（1）Object Names——对象名称

对象名称提供了一个简单的方法来共享对象句柄。一旦一个进程创建了一个命名事件对象、互斥量对象、信号量对象或者共享内存对象，其他进程可以使用这些对象名称以调用适当的函数来得到相应的对象句柄。

（2）Shared Memory——共享内存

RTSS 共享内存对象是一块空白的物理内存，可以映射入一个进程的虚拟地址空间。当一个共享内存对象被指定一个名称，其他的进程可以映射到这块内存。一个共享内存对象需要用一个句柄和一个虚拟地址来进行访问。为了解除对一个共享内存对象的访问，该进程应该关闭对象的句柄并解除对其虚拟地址的映射。当所有的进程都解除了对一共享内存对象的访问，这块内存返回到空白内存状态或者对象消失。

（3）Semaphores——信号量

RTSS 信号量对象是一个同步对象，它保持一个在 0 和某个最大值之间的计数值；这个计数值当一个线程完成对信号量对象等待的任务后减 1；当一个信号量释放时计数值增加一个变量值；当你申请一个 RTSS 信号量资源时，操作系统会检查是否有可用资源，如果有的话将计数减 1 以免让其他进程干预；仅仅当资源计数值减一之后，系统才让另一个线程请求一个资源。

（4）Event Objects——事件对象

事件对象是一个同步对象，它的状态可以用 RtSetEvent 或者 RtPulseEvent 来明确地设置。事件对象用来发送一个信号到一个线程非常有用，指示着一个特定的事件发生了。

（5）Mutex Objects——互斥对象

RTSS 互斥对象是一个同步对象，当它没有被任何一个进程用时为信号态而当它被一

个线程用时为非信号态。互斥对象对共享资源进行唯一访问权仲裁。通过互斥对象进行进程间的通信——为了同步在多个进程中运行的线程，包括 RTSS 进程和 Win32 进程，每一个进程中的线程一定有它自己的关于单个 RTSS 互斥对象的进程相关的句柄。

4.2.5 本控制系统涉及的编程接口（API）简介

下面将在开发本机器人伺服系统中用到的部分编程接口函数作一个简要的介绍[29]。

◆ 线程相关函数

RtCreateThread　　//创建线程
RtSuspendThread　　//挂起线程
RtResumeThread　　//使线程进入就绪态
RtSetThreadPriority　　//设置线程优先级
RtExitThread　　//退出线程
RtTerminateThread　　//终止线程

◆ 共享内存相关函数

RtCreateSharedMemory //创建共享内存区
RtOpenSharedMemory　　//打开共享内存区
Memcpy　　//读写共享内存区

◆ 进程间通信（IPC）相关函数

RtCreateMutex　　//创建互斥量同步对象
RtOpenMutex　　//打开互斥量对象
RtReleaseMutex　　//释放互斥量对象
RtCreateSemaphore　　//创建信号量同步对象
RtOpenSemaphore　　//打开信号量对象
RtCreateEvent　　//创建事件同步对象
RtOpenEvent　　//打开事件对象
RtSetEvent　　//触发某一事件对象
RtWaitForSingleObject　　//等待某一同步对象

◆ 时钟与计时器服务相关函数

RtCreateTimer　　//创建计时器并关联一个计时器线程
RtSetTimer　　//设定计时器值
RtSetTimerRelative　　//重新启动计时器
RtCancelTimer　　//取消计时器
RtSleepFt　　//让当前线程休眠一段时间

◆ I/O 设备相关函数

RtEnablePortIo　　//使能某个 I/O 设备

RtReadPortUchar	//读某个 I/O 设备（单字节）
RtWritePortUchar	//写某个 I/O 设备（单字节）
RtReadPortUshort	//读某个 I/O 设备（两字节）
RtWritePortUshort	//写某个 I/O 设备（两字节）
RtReadPortUlong	//读某个 I/O 设备（四字节）
RtWritePortUlong	//写某个 I/O 设备（四字节）

4.2.6 RTX 的实时性分析

RTSS子系统通过以下机制来保证实时性：

（1）实时子系统RTSS建立在RTX硬件抽象层扩展的基础上，因此具有直接对硬件操作的硬实时能力。

（2）RTSS 线程调度程序优先于所有 Windows NT 的调度程序，包括 Windows NT 管理的中断和延迟程序调度（DPCs）。这将使 RTSS 线程可以实时的完成预定任务。

（3）实时子系统同样支持进程间通信（IPC）， IPC 可以被 RTSS 或者 Win32 进程操纵。这使得实时和非实时程序的通讯和同步变得简单。

（4）RTX 硬件扩展层保持了中断相对于实时子系统和 Windows NT 的独立。Windows NT 系统不能屏蔽 RTSS 中断（在中断控制器层），Windows NT 中断在 RTSS 处理过程中被屏蔽。

（5）RTSS 为其进程提供其他时间要求严格的服务——如时钟和计时器。RTX 时钟被设计成保持一种与某种计时器周期同步的速率，也就是，此计时器已知在什么时候停止。这就可以计算计时器终止值和计时器滞后时间，从而消除 Windows NT 系统时钟相对于日时钟小度漂移的弊端。

从以上可以看出，RTSS实时系统通过直接操纵硬件的硬实时能力和线程优先调度、进程间通信、中断独立、高精度时钟等机制完美的解决了和Windows NT系统交互信息并实现Windows NT系统实时性的问题，使Windows NT+RTX既是一个通用操作系统，又是一个实时的操作系统。

4.3 本章小结

本章着重介绍了 Windows NT 操作系统以及相应的实时子系统 RTX 的体系结构、主要功能特性、各自及相互间的通信机制，阐述了 Windows NT 系统的在实时方面的缺陷和 RTX 系统在实时性上的优越性。应用 RTX 实时操作系统实现实时控制是实时控制的一个新的研究热点和研究方向，相信通过本章对 Windows NT 及 RTX 实时子系统的描述和介绍，读者对 Windows NT 及 RTX 会有一个比较全面的了解。

第五章　控制系统的软件设计

5.1　系统的控制流程

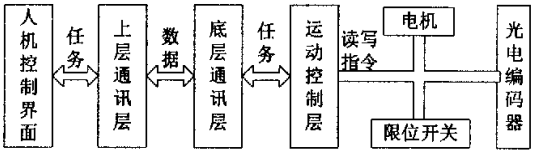


图 5-1 系统控制流程

机器人控制系统的软件设计采用友好的人机界面，机器人运动控制采用模块化设计，函数封装，通过人机界面控制底层实际上是调用不同的控制函数；函数的封装是采用动态链接库的方式，同时动态链接库包含 Win32 通信层，完成与底层实时子系统通信的任务；实时子系统接收来自上层界面的数据，然后进行运动控制，系统的控制流程如图 5-1 所示。控制任务首先由人机控制界面发出，Win32 通信层将这一任务转化为数据包传递给实时子系统的 RTSS 通信层，RTSS 通信层接收数据包后再转化成任务，最后运动控制层通过轨迹规划完成这一任务。

5.2　系统的软件设计

5.2.1　实时环境的架设

本文控制系统的开发使用 Visual C++6.0 软件，由于 VC 是微软的产品，用它来编写 Windows 程序有强大的程序接口和丰富的开发资源的支持，加之 VC 严谨的内存管理、在堆栈上良好的分配处理、生成代码的体积小，稳定性高的优点，所以 VC++就成为目前软件的主流开发工具。MFC 是 VC 的一个庞大的类库，包括丰富的构造人机界面的元素，控制系统人机界面就是采用 MFC 类库来进行开发；VC 也可以开发 Win32 环境下的应用程序，控制系统的函数封装和 Win32 通讯层就是通过开发 Win32 动态链接库来完成的，人机控制界面发出的任务通过调用 Win32 动态链接库的输出函数，如图 5-2 所示；RTX 实时环境也可以通过 VC 软件来实现，RTX 开发包可以嵌入 VC 的软件环境，控制系统的 RTSS 通信层和运动控制层是通过开发 RTSS 实时子系统来完成的，实时子系统的启动通过调用 Win32 动态链接库的函数[30,31]。

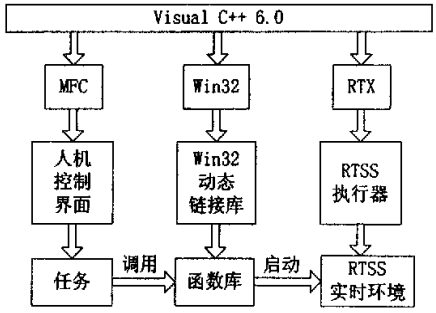


图 5-2 实时环境的架设

5.2.2 软件结构的描述

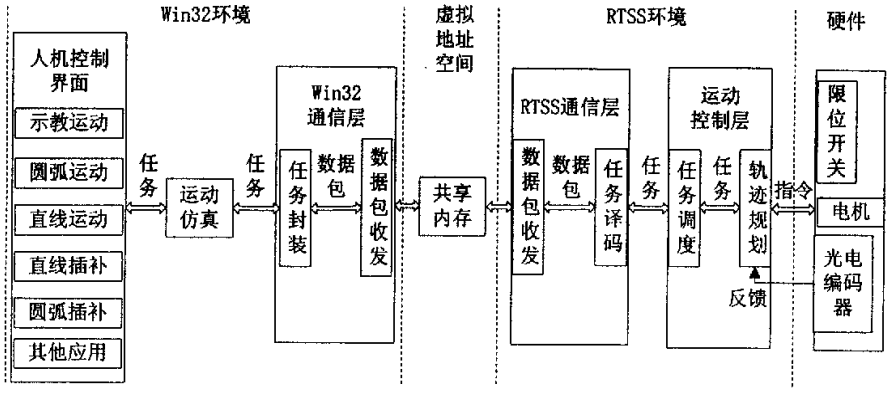


图 5-3 软件结构的描述

本控制系统软件结构如图 5-3 所示。总的来说，软件分为 Win32 环境和 RTSS 实时环境两大块，两大环境通过虚拟地址空间的共享内存机制进行通信。Win32 环境包括人机控制界面、运动仿真和 Win32 通信层。人机界面的任务先要进行运动仿真，以验证运动可行性，然后准备将任务传达到底层。任务传到底层具体由 Win32 通信层来完成，Win32 通信层首先将任务封装成数据包，以保证和 RTSS 子系统的有效数据传输，然后将数据包发送出去，即写共享内存。RTSS 子系统不断检测有无信息写入共享内存，一旦检测到，立即进行数据包接收工作，即读共享内存，接着进行任务译码，将数据包还原成任务，交给运动控制层执行。运动控制层首先进行任务调度，即识别某一具体任务，然后进行这一具体任务的轨迹规划，实现该任务。

5.2.3 软件用户界面及具备的功能

人机界面如图 5-4(a)，该界面集成了机器人运动仿真和机器人的运动控制。它分为三个区域：左边的操作区、右边的仿真区和顶上的菜单区。通过点击操作区的不同按钮可以进入不同的操作界面，进行操作可得相应仿真结果和机器人动作，如图 5-4(d)至 5-4(h)。

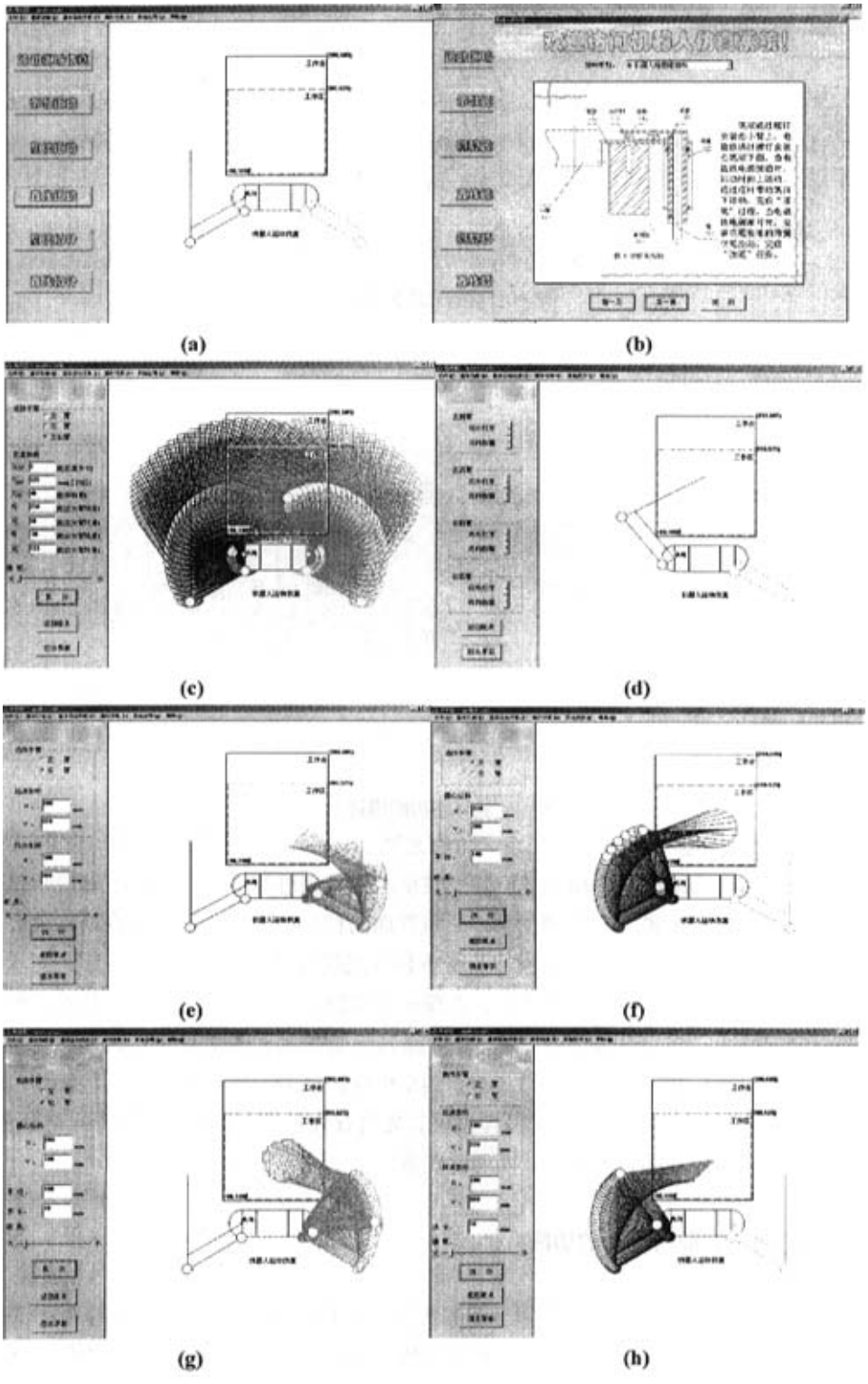


图 5-4 人机界面

工作区域仿真界面如图 5-4(c)，在这个界面可以进行工作区域仿真相关参数的设置并进行运动区域仿真，这些参数包括仿真手臂的选择、仿真步长、工作区纵坐标、平行四边形机构的传动角、左右臂摆动的具体范围及仿真速度。菜单区包括操作区的所有功能，此外它还包括机器人资料学习和整个界面相应控制功能，机器人资料学习界面如图 5-4(b)所示。

5.2.4 通信层及其实现

通信层包括 Win32 通信层和 RTSS 通信层，它们的实现方式基本相同，因此统一起来进行介绍。

5.2.4.1 通信层的构架

控制系统运行时，非实时任务进程与实时任务进程之间通过共享内存进行通信。为了传递控制命令和反馈信息，我们分别定义了两段共享内存，以构架通信的媒介，这两段共享内存格式按照数据编码和解码原理来定义。传递控制命令定义的共享内存格式如表 5-1，随后为相应的数据结构。

表 5-1 控制命令共享内存格式

起始部分	数据单元检索信息	数据校验	数据单元	单元校验	以下重复4～5 内容

```
typedef struct ORRCIPCBufHeader  //共享内存起始部分数据结构
{int nHeaderState;          //共享内存起始部分的状态
int nHeaderSize;            //共享内存起始部分的大小
int nUnitNum;               //共享内存数据单元的数量
int nUnitStartAddr;         ////共享内存数据单元的起始地址
}ORRCIPCBUFHEADER,*PORRCIPCBUFHEADER;
typedef struct ORRCIPCBufUnitInfo   //共享内存数据单元检索信息数据结构
{int nUnitID;               //单元号
int nAddrOffset;            //单元的偏移地址
int nBufTotalSize;          //单元总的大小
int nBufUnitMaxDataSize;                          //单元容量的最大值
int nUnitState;                                   //单元的状态
TCHAR szORRCIPCBufUnitDataValidEventName[128];    //单元的数据合法事件名
TCHAR szORRCIPCBufUnitIdleEventName[128];         //单元的空闲事件名
TCHAR szORRCIPCBufUnitAccessMutexName[128];       //单元的互斥事件名
}ORRCIPCBUFUNITINFO,*PORRCIPCBUFUNITINFO;
```

```
typedef struct ORRCIPCBufUnitHeader   //共享内存数据单元起始部分数据结构
{long lnBufUnitID;              //单元的代号
int nUnitState;                 //单元的读写状态
int nBufTotalSize;              //单元总的大小
int nBufUnitMaxDataSize;        //单元容量的最大值
int nBufUnitValidDataSize;      //单元容量的合法值
long lnSenderID;                //发送代号
}ORRCIPCBUFUNITHEADER,*PORRCIPCBUFUNITHEADER;
```

以上数据结构参数的意义可以参考其名字和行后注释。传输时数据单元的内容是这样的：命令头+任务段，命令头的数据结构为：

```
typedef struct CommandDataHeader
{int nCommandCode;      //任务代号
int nParamSize;         //任务包的大小
int nPriorty;           //任务的优先级
int nUnitState;         //共享内存缓冲区的读写状态
long lnSenderID;        //发送代号
}COMMAND_DATA_HEADER,* PCOMMAND_DATA_HEADER;
```

其中，任务代号有不同种类，我们用宏的方式定义了与其对应的任务格式，比如CC_QUIT、CC_LINE、CC_CIRCLE、CC_ARM_CONTROL、CC_AXIS_MOVE、CC_GET_CUR_POS、CC_HOME、CC_GRASPER 等。

任务段的数据结构根据以上任务格式有相应的数据结构，比如右臂绘图笔动作和圆弧运动指令就不同，下面就圆弧运动指令所需的数据结构来说明：

```
typedef struct ROBOTCIRLE     //机器人圆弧运动的数据结构
{   MyPoint     center_point;   //圆心坐标
    double      radius;         //半径
    double      start_angle;    //圆弧运动起始角度
    double      end_angle;      //圆弧运动终止角度
    Info    info;               //运动信息（包括速度、运动状态、任务误差等）
} RightArm_circle,LeftArm_circle,*PRightArm_circle;
```

反馈信息共享内存定义的格式如表 5-2，它定义的数据结构类似以上构造。

表 5-2 反馈信息共享内存格式

起始部分	单元数字量	重复 2 的内容	单元模拟量	重复 4 的内容	数据长度

以上两段共享内存的数据保护通过采用互斥和临界区机制来完成，具体如下图 5-5 所示：

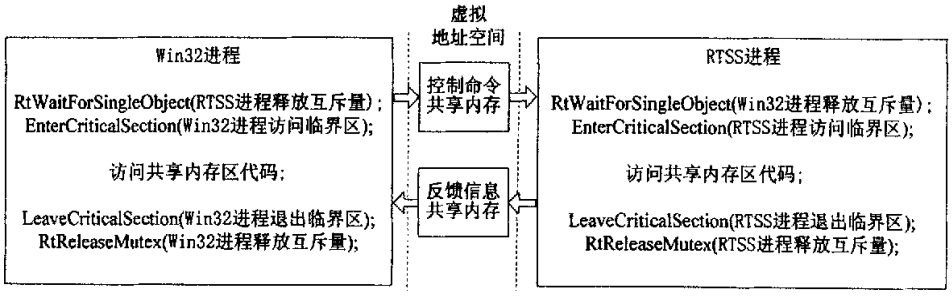


图 5-5 共享内存数据保护机制

从图中可以看出 Win32 进程和 RTSS 进程不能同时访问共享内存，必须等待另一个进程访问完毕释放互斥量时才能访问共享内存，从而有效地保护了数据。

5.2.4.2 通信层的模块实现

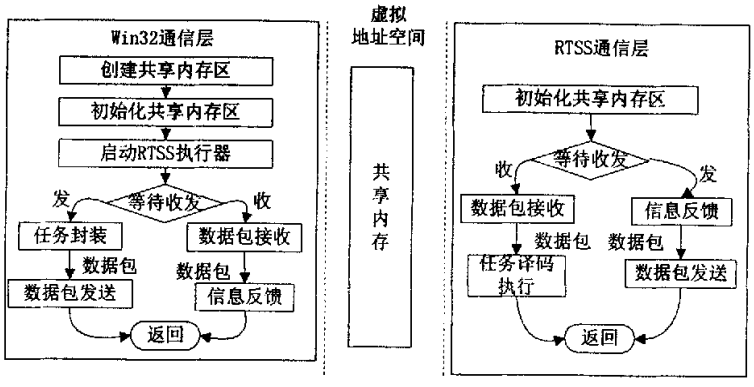


图 5-6 通信层模块流程图

通信层功能模块如图 5-6 所示，Win32 通信层和 RTSS 通信层都要进行共享内存的初始化和数据包的收发，所不同的是 Win32 通信层首先要创建共享内存区并启动 RTSS 执行器，以便启动 RTSS 通信层。创建共享内存区主要包括以下内容：

（1）调用创建共享内存区函数，以获得共享的虚拟地址空间和其句柄；

（2）创建一系列通信事件和互斥量，以使通信线程能够很好的同步；

（3）初始化一系列通信全局变量。

初始化共享内存区主要包括一下内容：

（1）打开共享内存通信区，把这个已存在的共享内存对象映射入自己的虚拟地址空间；

（2）打开通信事件和通信互斥量，以保证通信的正常进行；

通信层信息的收发是通过调用 SendTo（）和 GetData（）这两个通信接口函数来完成的。

（1）数据包发送接口函数：

SendTo(long lnSrcCommID,long lnDesCommID,BYTE * pData,int nDataSize,int nTimeout)

其中参数包括发送端通信 ID 号 lnSrcCommID，接收端通信 ID 号 lnDesCommID，发送缓冲区指针 pData，发送数据大小 nDataSize，命令返回等待时间 Timeout。

接口函数主要内容有：

a) 共享内存初始部分地址赋值于 pIPCBufHeader，并等待访问共享内存起始部分数据有效事件 hORRCIPCBufHeaderValidEvent；

b) 数据单元检索信息地址赋值于 pIPCBufUnitInfo，并等待数据访问互斥量 hORRCIPCBufAccessMutex；

c) 根据接收端通信 ID（lnDesCommID）和数据单元检索信息中的偏置信息（nAddrOffset）定位数据单元头部地址，并定位数据单元有效数据段的地址；

d) 打开数据单元空闲事件（szORRCIPCBufUnitIdleEventName）并随后等待，打开数据单元有效事件（szORRCIPCBufUnitDataValidEventName）并随后等待，打开数据单元访问互斥量（szORRCIPCBufUnitAccessMutexName）并随后等待；

e) 如果发送数据大小（nDataSize）大于数据单元的最大数据容量，则直接返回；反之，将要发送的数据赋给共享内存中相应的数据单元有效数据段；

f) 重置访问数据单元空闲事件（szORRCIPCBufUnitIdleEventName），设置访问数据单元有效事件（hORRCIPCBufHeaderValidEvent），释放数据单元访问互斥量（szORRCIPCBufUnitAccessMutexName）。

（2）数据包接收接口函数：

GetData(long　lnCommBufID,void * pDataBuf,int nBufSize,int * pDataSize,int nTimeout)

其中参数包括接收端通信 ID 号 lnDesCommID，接收缓冲区指针 pData，接收缓冲区大小 nBufSize，接收数据大小 nDataSize，命令返回等待时间 Timeout。

接口函数主要内容有：

a) 等待数据单元访问互斥量 hORRCIPCBufUnitInfoAccessMutex；

b) 根据接收端通信 ID（lnCommBufID）得到数据单元的访问指针（以局部数据单元的形式表示）；

c) 待数据单元有效事件（szORRCIPCBufUnitDataValidEventName），等待数据单元空闲事件（szORRCIPCBufUnitIdleEventName），等待数据单元访问互斥量（szORRCIPCBufUnitAccessMutexName）；

d) 若指定接收数据大小大于数据单元的有效数据量，直接将数据复制给接收缓冲区指针；并将接收到的数据大小赋给 pDataSize，并重置访问数据单元有效事件（hORRCIPCBufUnitDataValidEvent），设置访问数据单元空闲事件（szORRCIPCBufUnitIdleEventName），并释放数据单元访问互斥量（szORRCIPCBufUnitAccessMutexName），此过程与 SendTo 中的第 f)步是完全相对应的；

e) 若指定接收数据大小小于数据单元的有效数据量，则先接收 nBufSize 大小的数据量，并将后续数据左移，并进入第 d)步和第 e)步的判断，直到满足第 d)步的要求。

5.2.5 运动控制层及其实现

运动控制层主要完成上层传输下来的任务，通过各种轨迹规划算法实现不同的任务。由于机器人运动对实时性的要求比较高，该层在 RTSS 实时环境下实现。

5.2.5.1 运动控制层的数据结构

为了便于理解下文的程序流程图和相关讲解，本节给出运动控制层所需的数据结构。它专门为机器人右臂的伺服系统设计的，同时兼顾左臂，数据结构中各个变量的意义见旁边注释。

```
typedef struct   POINT   //机器人手臂关节点的定义
{   double  x;
    double  y;
}MyPoint;
typedef struct   ANGLE  //机器人手臂关节角的定义
{   double      f;   //小臂关节角
    double      b;   //大臂关节角
}ArmAngle;
typedef struct   MOVESTATUS   //机器人运动状态的定义
{   int  is_final;      //指示任务是否进行到最后一步
    int  is_start;      //指示任务是否开始
    int is_new;         //指示任务是否与前一次任务相关
}MoveStatus;
typedef struct   TASKERROR     //机器人的任务误差的定义
{   long int front_Angle_error;   //小臂角度误差
    long int back_Angle_error;   //大臂角度误差
}TaskError;
typedef struct   INFO          //机器人运动信息的定义
{int          speed_level;     //运动速度
TaskError    error;           //机器人的任务误差
MoveStatus  move;             //机器人的运动状态
}Info;
typedef struct   ROBOTVOLT     //伺服电机控制电压
{   double  f_v;     //小臂电机控制电压
    double  b_v;     //右大臂电机控制电压
    int  min_v;      //电机控制电压最小值
    int  max_v;      //电机控制电压最大值
```

```
}RobotVolt;
typedef struct   POINTARMANGLE  //机器人正解和反解的数据结构
{   int          armNO;         //手臂号，1—左臂，2—右臂
    MyPoint      ending;        //小臂末端位置
    MyPoint      joint;         //大臂末端位置
    double       mFrontAngle;   //小臂角度
    double       mBackAngle;    //大臂角度
}PointToArmAngle,ArmAngleToPoint;
typedef struct RIGHTANGLEMOTION    //机器人右臂摆动的数据结构
{   int          target_ERROR;      //目标误差
    ArmAngle     front_back;        //大小臂摆动角度
    RobotVolt    voltage;           //控制电压
    Info     info;                  //运动信息
}RightAngleMotion,*PRightAngleMotion;
typedef struct LEFTANGLEMOTION     //机器人左臂摆动的数据结构
{   ArmAngle     front_back;        //大小臂摆动角度
    Info     info;                  //运动信息
}LeftAngleMotion,*PLeftAngleMotion;
typedef struct ROBOTLINE           //机器人直线运动的数据结构
{   MyPoint      start_point;       //运动起点
    MyPoint      end_point;         //运动终点
    Info     info;
} RightArm_line,LeftArm_line,*PRightArm_line;
```

圆弧运动数据结构在 5.2.4.1 节已定义，此处略去。

5.2.5.1 运动控制层的模块实现

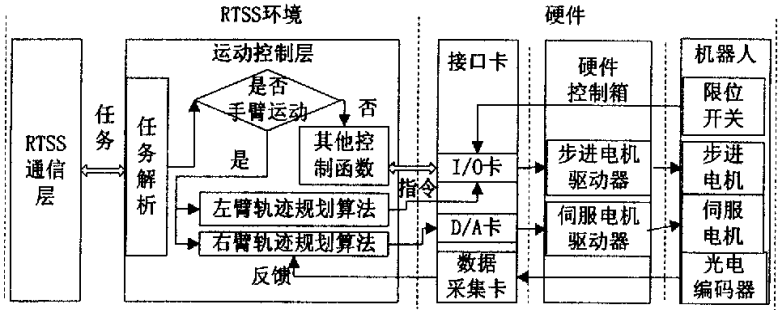


图 5-7 运动控制层模块

运动控制层的模块实现见图 5-7，由于该模块与硬件紧密联系，图上给出了硬件的详

细资料。从图上可以看出，运动控制层完成任务的调度和各种运动控制的实现，任务解析主要将任务分类及各变量初始化，各种运动控制包括右臂绘图笔落下和抬起、左臂棋抓上升和下移、左臂棋抓抓放棋子、左右臂回零、左右臂手臂摆动。其中，左右臂手臂摆动要用到轨迹规划算法，其他运动可以简单实现。右臂的轨迹规划算法将在下章重点介绍。下面以右臂运动为例说明各种板卡是如何进行操作的。

（1） I/0 卡

I/O 卡主要应用在读限位开关、绘图笔抬起和落下操作中。读限位开关使用的函数为：

```
RtEnablePortIo(IO_Card_Base,IO_Card_Range);  //使能 I/O 卡
Reads=RtReadPortUchar(IO_Card_Base); // IO_Card_Base 为限位开关 I/O 地址
```

绘图笔动作对应的函数为：

```
RtWritePortUchar(IO_Card_Base+2,ControlPort102);
```

其中，IO_Card_Base 是 I/O 卡的基地址，2 表示绘图笔电磁铁开关地址对基地址的偏移量，ControlPort102 表示写端口变量。

（2） D/A 卡

D/A 卡主要用来给定伺服电机的控制电压，对其操作函数有：

```
RtEnablePortIo(DA_Card_Base,8);          //使能 D/A 卡
RtWritePortUchar(DA_Card_Base+5,front_u); // 给定右小臂伺服电机控制电压
RtWritePortUchar(DA_Card_Base+6,back_u); // 给定右大臂伺服电机控制电压
```

其中，DA_Card_Base 为 D/A 卡基地址，常数 5、6 分别表示右小臂、右大臂控制端口相对于基地址的偏移量，front_u、back_u 为电压数字给定变量

（3） 数据采集卡

数据采集卡主要完成编码器脉冲的采集和计算，对其操作的函数有：

```
ENC6_REGISTRATION(ENC300_Card_One,ENC300_Card_Base);//使能数据采集卡
ENC6_INIT_CARD(ENC300_Card_One,0,0,0,0,0,0); //初始化数据采集卡
ENC6_RESET_ENCODER(ENC300_Card_One, X1_axis); //初始化卡的读数
ENC6_RESET_ENCODER(ENC300_Card_One, X2_axis);
mPulses_front=ENC6_GET_ENCODER(ENC300_Card_One,X1_axis);//读卡值
mPulses_back=ENC6_GET_ENCODER(ENC300_Card_One,X2_axis);
```

可以看出，以上操作数据采集卡的函数没有采用 RTX 函数，下章将详细说明这些函数是如何用 RTX 函数实现的。

5.3 实时系统的编译

图 5-2 说明实时系统是如何架设的，实时系统的编译可以通过更改图 5-2 得到图 5-8 来进行说明。从图中可以看出，人机界面机器人仿真应用程序链接入 Win32 动态链接库

的输出库头文件 MFCLibExp.h、库文件 MFCLib.lib 和动态链接文件 MFCLib.dll，同时加入 RTX 应用程序的输出的执行器 RTController.rtss，编译运行后可以形成人机界面机器人控制程序。其中，Win32 动态链接库在编译时要链接入 rtapi_w32.lib 库文件，以便调用 RTX 环境下的函数；RTX 应用程序在编译的时候要链接入 startupCRT.obj、RTXlibcmt.lib、oldnames.lib、rtapi_rtss.lib、rtx_rtss.lib、w32_dll.lib 等目标文件和库文件，以便实现 RTX 的相关实时机制。

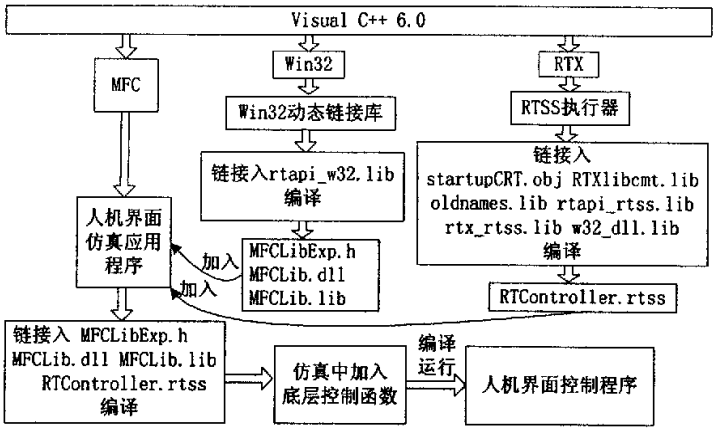


图 5-8 软件系统的编译

5.4 本章小结

本章重点讲述了机器人控制系统的软件设计。首先介绍了系统的控制流程，展示了系统的大体框架和命令数据的流向；接着逐步深入地论述了软件系统的实时环境、软件结构、软件用户界面、软件系统的通信层和运动控制层；最后说明了软件系统是如何编译的。

第六章　机器人的伺服控制研究

6.1　伺服控制建模

6.1.1 系统组成原理

位置伺服系统框图见图 6-1，系统位置环由上位机、伺服驱动器和交流伺服电机 M 组成。其中，上位机是工控机，光电编码器固连在伺服电机上。工控机通过数据采集卡来收集电机的角位移信号，主要完成对采集脉冲信息的处理，并通过某种算法输出控制信号，经过 D/A 卡转变成模拟电压，然后通过伺服驱动器来驱动电机运行。

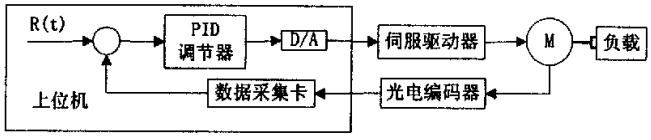


图 6-1 位置伺服系统框图

6.1.2 频率响应实验

对伺服控制系统进行设计的前提是建立该伺服系统的模型和控制算法，系统模型采用频率响应法建立，控制算法采用 PID 算法。为了辨识模型参数，设计了频率响应实验，实验步骤如下：

（1） 获取实验数据

表 6-1 伺服驱动器参数设置

参数号	值	意义
02	1	选择速度控制方式
11	200	第一速度环增益
12	6	第一速度环积分时间常数
50	60	速度指令输入增益，使伺服电机速度最大值达到 900 度/秒

首先把伺服驱动器设置成速度控制方式，在速度控制方式下，期望控制电压与电机转速成正比，故参数的调节目的是使控制电压尽可能的与转速成正比，具体参数设置见表 6-1，其他参数取系统默认值即可。

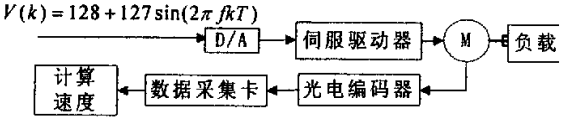


图 6-2 频率响应法试验框图

频率响应试验框图见图 6-2，图中，V(k)是正弦输入电压激励信号，f 是激励信号的

频率，一般设置在 0.1Hz 和 10Hz 之间，T 为采样周期，为了得到准确的伺服电机转速，T 定为 1ms。

为了保证正弦控制电压信号的实时产生，程序必须在实时子系统 RTSS 环境下运行，并采用 1us 精度的 RTX 计时器，使发送电压指令和采集脉冲信号以相同的时间间隔进行，并将计算得到的速度值不断存入文件中。

（2） 实验数据滤波

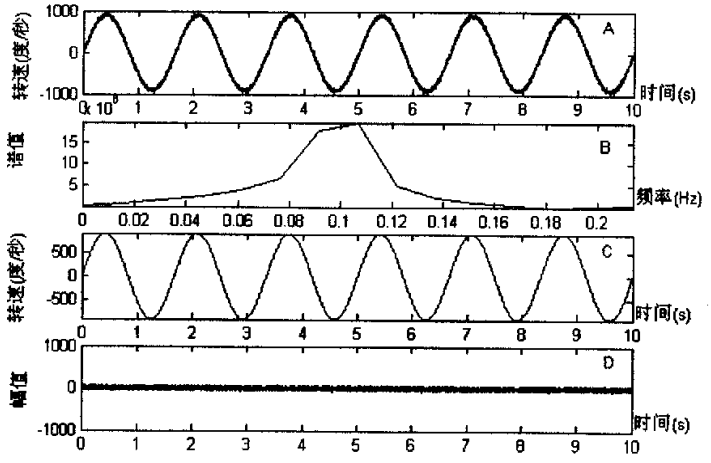


图 6-3 右小臂 0.1Hz 正弦激励电压时电机转速采样数据分析

由于在实验过程中，不可避免的存在一些外部干扰，造成一些高频分量，这些高频成分可以从右小臂正弦激励电压下电机的转速采样数据看出，如图 6-3 中子图 A，相应频谱图如子图 B。选用 n 阶低通巴特奥斯滤波器，由于数据以 1000Hz 的频率采集，设定各个参数如下：通带范围为$[0, f+1.6]$，其纹波系数为 3，阻带范围为$[f+100, 500]$（500 是奈奎斯特频率），且以 60dB 衰减[32,33]。滤波后的信号如子图 C，滤掉的干扰信号如子图 D。

（3） 实验数据分析

设待辨识系统的传递函数为 G(s)，幅频特性可以表示为：

$$M(w)=\frac{W_{\max}(w)}{2.5}=|G(w)| \qquad (6\text{-}1)$$

其对数幅频特性为：　　$L(w)=20\log(M(w))$　　(6-2)

其中，$W_{\max}$为转速的最大值，$w=2\pi f$ 代表圆频率。

运用 MATLAB 工具对存于文件中的数据进行滤波、分析可以求出 $W_{\max}$ 的值，也就是表 6-2 中右小臂频率响应特性转速的峰值。表 6-2 中峰峰值是转速最大值和最小值的绝对值之和。幅频特性值和对数幅频特性值分别见表 6-2 第 4 列和第 5 列。采用一次拟合曲线绘制的幅频特性曲线见图 6-4，小圆圈形成的曲线来源于实验所得的数据，光滑曲线是一次拟合曲线。右小臂的拟合曲线为：

$$L(w)=-0.0937w+51.2976 \qquad (6\text{-}3)$$

表 6-2 右小臂速度频率响应特性数据

频率(Hz)	峰峰值（度/秒）	峰值（度/秒）	幅频特性值	对数幅频特性值（DB）
0.1	1.8113528e+003	905.6764	362.2706	51.1807
0.2	1.8134979e+003	906.7489	362.6996	51.1909
0.3	1.8148084e+003	907.4042	362.9617	51.1972
0.4	1.8091457e+003	904.5728	361.8291	51.1701
0.5	1.8113666e+003	905.6833	362.2733	51.1807
0.6	1.8113088e+003	905.6544	362.2618	51.1804
0.7	1.8125776e+003	906.2888	362.5155	51.1865
0.8	1.8125058e+003	906.2529	362.5012	51.1862
0.9	1.8026398e+003	901.3199	360.5280	51.1388
1	1.8123990e+003	906.1995	362.4798	51.1857
2	1.8142214e+003	907.1107	362.8443	51.1944
3	1.8090099e+003	904.5049	361.8020	51.1694
4	1.8194790e+003	909.7395	363.8958	51.2195
5	1.7990059e+003	899.5030	359.8012	51.1213
6	1.7661085e+003	883.0542	353.2217	50.9609
7	1.7338091e+003	866.9046	346.7618	50.8006
8	1.6946862e+003	847.3431	338.9372	50.6024
9	1.6383481e+003	819.1740	327.6696	50.3087
10	1.5637772e+003	781.8886	312.7554	49.9041

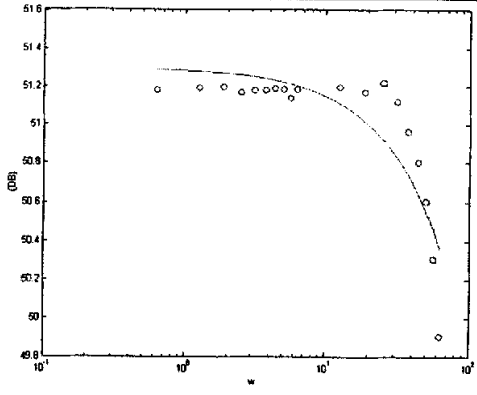


图 6-4 右小臂对数幅频曲线

同理可以求出右大臂的图表数据，形成表 6-3 和图 6-5。

（4） 模型求解。

被辨识对象可以简化成一阶惯性系统[34]，其传递函数为：

$$G(s)=\frac{1}{Js+B} \tag{6-4}$$

其转折频率 $w_c=B/J$，则可得其低频特性：

$$L(w)=-20\lg B,\ w<<w_c \tag{6-5}$$

表 6-3 右大臂速度频率响应特性数据

频率(Hz)	峰峰值（度/秒）	峰值（度/秒）	幅频特性值	对数幅频特性值（DB）
0.1	1.8256787e+003	912.8394	365.1357	51.2491
0.2	1.8205845e+003	910.2922	364.1169	51.2248
0.3	1.8206204e+003	910.3102	364.1241	51.2250
0.4	1.8137247e+003	906.8623	362.7449	51.1920
0.5	1.8154912e+003	907.7456	363.0982	51.2005
0.6	1.8154526e+003	907.7263	363.0905	51.2003
0.7	1.8139653e+003	906.9826	362.7931	51.1932
0.8	1.8077313e+003	903.8657	361.5463	51.1633
0.9	1.8163622e+003	908.1811	363.2724	51.2046
1	1.8248706e+003	912.4353	364.9741	51.2452
2	1.8290953e+003	914.5476	365.8191	51.2653
3	1.8134615e+003	906.7308	362.6923	51.1908
4	1.8311346e+003	915.5673	366.2269	51.2750
5	1.8206026e+003	910.3013	364.1205	51.2249
6	1.8176680e+003	908.8340	363.5336	51.2109
7	1.7982453e+003	899.1226	359.6491	51.1176
8	1.7583314e+003	879.1657	351.6663	50.9226
9	1.6776460e+003	838.8230	335.5292	50.5146
10	1.6044077e+003	802.2039	320.8815	50.1269

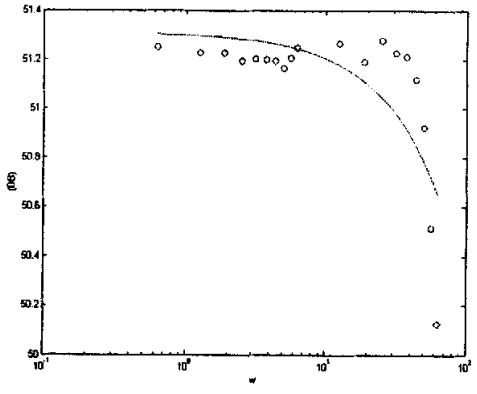


图 6-5 右大臂对数幅频曲线

实验曲线和理论曲线在低频段的值应该相等，故有：

$-20\lg B=51.2976$，　　得 $B=0.0027v$/deg/s

由一阶惯性系统对数幅频特性曲线得特性可得[35]：

$-0.0937w_c=-3$dB，　　得 $w_c=163.5$rad/s

故有：　　$J=B/w_c=8.5063e\text{-}005\ v/deg/s^2$

采用同样得方法求得右大臂的模型数据为：

$B=0.0027v$/deg/s　　$J=5.9658e\text{-}005\ v/deg/s^2$

6.2　目标轨迹规划算法

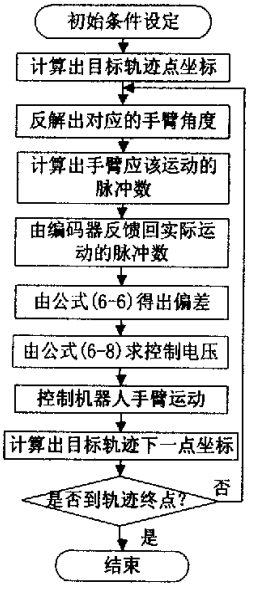


图 6-6 目标轨迹规划算法

这种方法先求出目标轨迹 $\theta(t)$，然后根据每一时刻 $\theta(t)$和反馈轨迹 $\theta_k(t)$的偏差 $e(t)$来控制机器人的运动[36]。偏差 $e(t)$由公式（6-6）求得，目标轨迹规划算法见图 6-6。

$$e(t)=\theta_k(t+1)-\theta(t) \qquad (6\text{-}6)$$

其中，t=N*T，N=0,1,2,3……

我们采用 PID 控制算法来控制偏差，控制目标快速的缩小当前偏差。PID 算法的主要公式如下：

$$U(k)=K_p e(kT)+K_i\sum_{i=0}^{n}e(i)+K_d(e(k)-e(k-1)) \qquad (6\text{-}7)$$

其中，$U(k)$为控制电压，K_p 是比例系数，K_i 是积分系数，K_d 是微分系数，T 为采样周期。

比例系数 Kp 将控制系统的偏差成比例地反映到控制电压 $U(k)$上，偏差一旦产生，控制器立即产生控制作用，以减少偏差。Kp 增大，使伺服系统的动作灵敏，响应加快；积分环节主要用于消除系统稳态误差；微分环节加快系统的动作速度，减少调节时间。采用 PID 控制时去除积分项和微分项，这是因为：

（1）公式（6-6）中 $\theta_k(t+1)$是有规律的目标轨迹函数，$\theta(t)$是在各种因素干扰下形成的实际轨迹，由编码器的脉冲反馈求得。因此，偏差 $e(t)$没有一定的规律，且相邻时刻的偏差没有相关性，没有必要积累偏差。

（2）确切的说，积分系数 K_i 是用来消除控制过程稳态时的静差。但是，PID 控制的目的是快速的响应并缩小当前偏差而不是消除偏差，如果偏差消除了机器人的运动也就

停止了，即机器人的运动是通过偏差来驱动的。因此，积分系数 K_i 必须为 0。

（3）公式（6-6）中 $e(t)$往往比较小，采用微分参数后容易引起控制电压波动太大，以致改变电压的极性，从而引起振动。在实验过程中已发现这种弊端，故将微分项去掉。采用 PID 控制时只采用比例项，使得手臂运动时能快速的响应误差并减少误差，对应 PID 算法：

$$U(k)=K_p e(kT) \qquad (6\text{-}8)$$

比例因子采用 MATLAB 工具来求解，用 rlocus 函数可以画出上述根轨迹图。如图 6-7 根轨迹曲线所示，C 为原点，当根轨迹增益从零变到无穷时，相应根轨迹点从 A(D)点运动到 B(C)点，不会穿越虚轴进入右半平面，因而系统是稳定的。处于 O 点时，系统为临界阻尼系统，单位阶跃响应速度较 AO 和 CO 段快；处于 OD 和 OB 段时，系统为欠阻尼系统，单位阶跃响应的超调量将随根轨迹增益的增大而增大。因而，比例因子取 O 点的根轨迹增益。

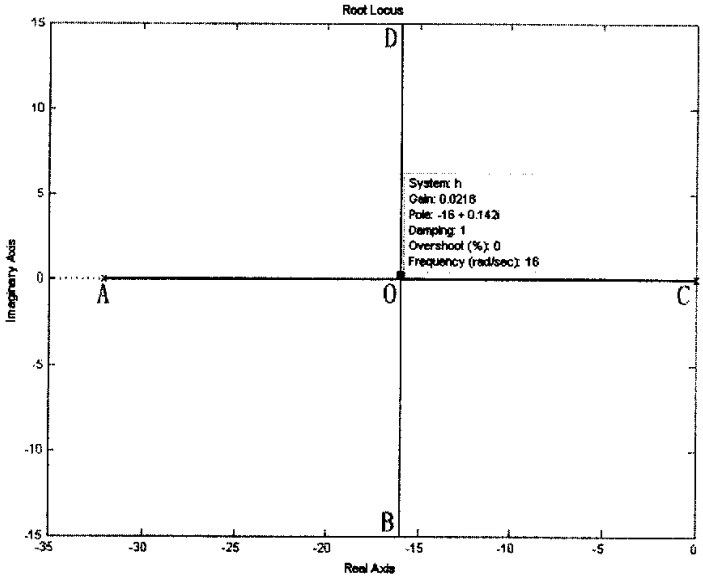


图 6-7 右小臂根轨迹曲线

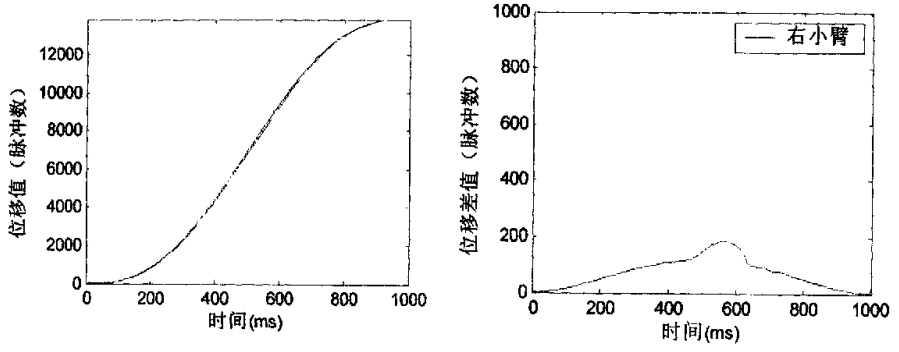
6.2.1 单关节实验

单关节实验是进行机器人双关节轨迹运动重要基础，应用单关节实验，我们不仅可以验证上述 PID 算法的正确性，而且能够确定目标轨迹规划算法适用的速度范围。试验时选用经典的 5 阶多项式曲线作为输入曲线：

$$\theta_r = A\left[6\left(\frac{t}{T_r}\right)^5 - 15\left(\frac{t}{T_r}\right)^4 + 10\left(\frac{t}{T_r}\right)^3\right] \qquad (6\text{-}9)$$

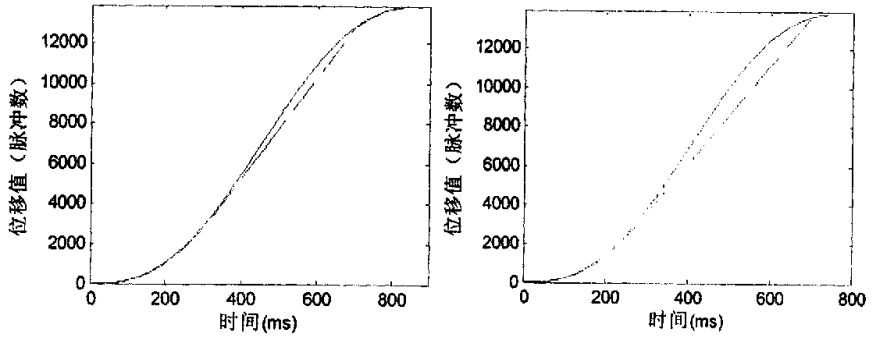
式中，A 是幅值，T_r是上升时间。在该次实验过程中幅值 A 为 10 度，转化为脉冲数为 13889 个脉冲。因为该曲线的 3 次微分在起点和终点处仍然是光滑连续的，其可以避

免伺服驱动系统和被控对象受高频动力学响应的冲击。

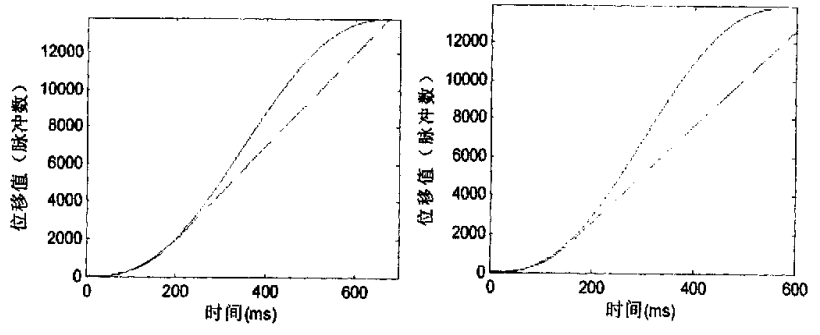


(a)　　(b)

图 6-8 单关节实验曲线



(a) (T_r=0.9s)　　(b) (T_r=0.8s)



(c) (T_r=0.7s)　　(d) (T_r=0.6s)

图 6-9 其他单关节实验曲线

图 6-8(a)是右小臂单关节摆动角位移曲线，为了计量准确，位移采用脉冲数为单位，时间采用毫秒为单位。图中实线为目标位移曲线，虚线为实际手臂运动位移曲线，运动时间 T_r 为 1s。图 6-8(b)是相应的位移差值曲线，如图中所示。横坐标中间部位差值的幅度最大，且为 186 个脉冲数。已知伺服电机到手臂的减速比为 50，手臂摆动 1 度对应 1388.89 个脉冲，右小臂长 435mm，故最大误差转换为伺服电机的弧度误差为 186/1388.89*PI/180＝0.0023rad，其中，PI 表示圆周率。折合到右小臂的摆动弧度误差为

0.0023/50＝4.6747e-005 rad，相应手臂末端位置误差为 4.6747e-005*435＝0.0203mm。可以看出该误差是相当小的，适用于高精度的运动控制。手臂末端速度为 435*10*PI/180/1＝75.9218mm/s。

我们逐步缩小运动时间 T_r，可以得到 T_r 为 0.9s、0.8s、0.7s、0.6s 时的单关节运动曲线，其对应的右小臂的最大位置误差分别为：0.0780mm、0.1628mm、0.2672mm、0.3929mm，如图 6-9 中(a)、(b)、(c)、(d)所示。对应末端速度分别为 84.3576mm/s、94.9023mm/s、108.4597mm/s、126.5364 mm/s。可以看出，T_r＝0.7s 时最大位置误差已经达到 0.2672mm，不适宜进行高精度的位移控制，T_r＝0.6s 时手臂末端运动终点已出现位置误差且为 1211 脉冲，已不能满足伺服控制的需要。因此，右小臂末端速度应限制在 108.4597mm/s 以下。

6.2.2 双关节应用

我们以圆弧轨迹运动为例来介绍目标轨迹规划法在双关节运动中的应用，圆弧半径 50mm，其控制流程图见图 6-10(a)，末端轨迹如图 6-10(b)所示。从图 6-10(b)中可以看出，圆弧轨迹没有明显纹波，表明运用目标轨迹规划法进行双关节运动跟踪目标轨迹的能力比较强；圆弧末端的封闭性比较好，说明双关节运动几乎没有位置误差。圆弧运动耗时 T_r 为 13.688s，得手臂末端平均运动速度 2*PI*50/13.688＝22.9514mm/s，PI 为圆周率。

图 6-11(a)是圆弧轨迹运动中电压值的变化曲线，从图中可以看出，电压的波动范围很小（电压值 128 处代表 0 伏），右小臂在 120 和 137 之间变化，右大臂在 114 和 141 之间变化，电压值的允许范围为 0-255，表明该次运动中电机的速度比较低。

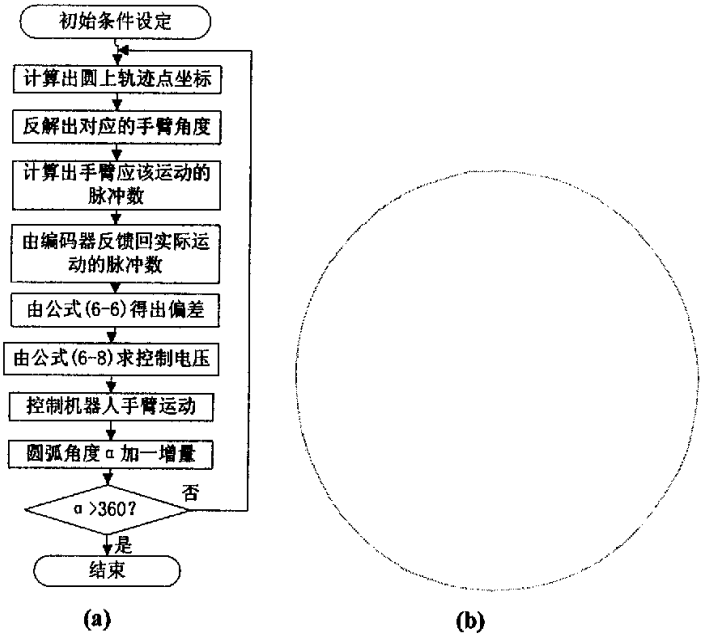


图 6-10 目标轨迹规划算法及运行结果

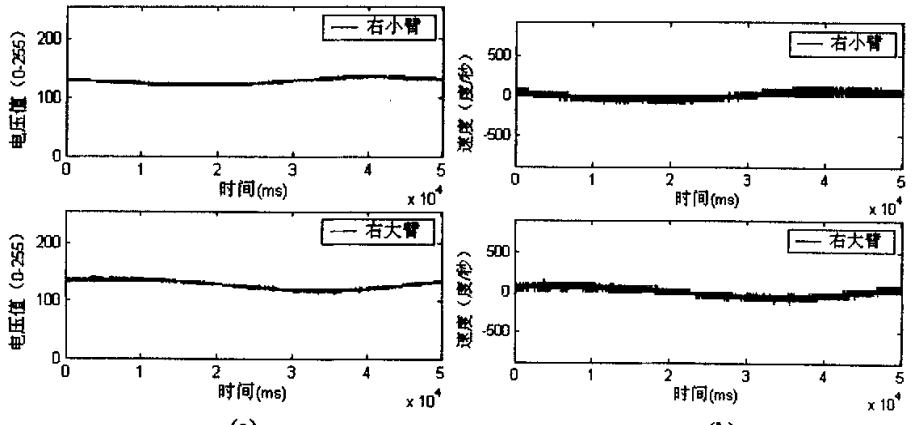


图 6-11 轨迹运动中电压值和速度的变化

图 6-11(b)是圆弧轨迹运动中速度值的变化曲线，右小臂速度在-108 度/秒和 108 度/秒之间变化，右大臂速度在-108 度/秒和 144 度/秒之间变化，速度值的允许范围为-900 —900 度/秒。对比以上两图可以看出，速度的波动和电压的波动大致相似，即速度和电压近似成正比。

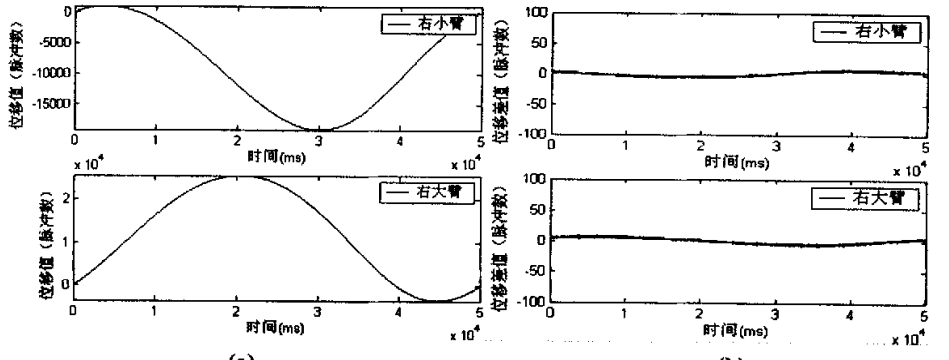


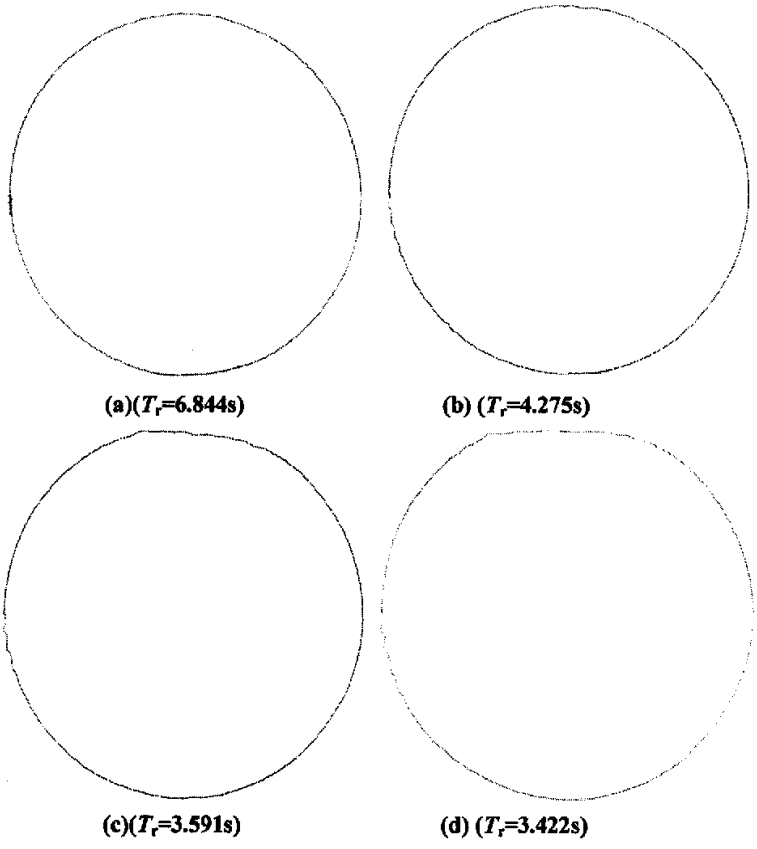
图 6-12 目标位移和位移偏差曲线

图 6-12(a)是圆弧轨迹运动中的目标位移曲线，图 6-12(b)是相应的目标位移和实际位移的位移偏差曲线，右小臂位移偏差在 8 和-6 之间变化，右大臂位移偏差在 8 和-8 之间变化，最大偏差值 8 个脉冲数，折合到手臂末端的位置误差约 8.7462e-004mm，说明 PID 控制算法取得了良好的效果。

逐步缩短圆弧运动时间，得到图 6-13 所示轨迹曲线。图 6-13(a)、(b)、(c)、(d)对应曲线的手臂末端平均速度为 45.9029 mm/s、73.4875 mm/s、87.4852 mm/s、91.8057 mm/s。可以看出当末端速度为 87.4852 mm/s 时圆弧曲线已经发生轻微畸变，其手臂末端的位置误差约 0.619mm。当末端速度为 91.8057 mm/s 时圆弧曲线发生严重畸变，其手臂末端的位置误差约 1.204mm，故圆弧运动时速度应小于 87.4852 mm/s。同时可以看到，速度逐渐提高时圆弧运动起始处已经出现纹波。

目标轨迹规划算法类似于插补算法，依据反馈位置和目标位置的偏差的正负判断运

动方向，依据反馈位置和目标位置的偏差的大小来控制运动速度的幅度。有区别的是插补运动用计算替代反馈，来估计控制对象的运动位置，然后再和目标轨迹相比较，判断进行下一步运动的方向；而且参与插补运动的运动轴相互垂直，插补运动可以分解为两个坐标轴的直线运动。



(a)(T_r=6.844s)　(b) (T_r=4.275s)

(c)(T_r=3.591s)　(d) (T_r=3.422s)

图 6-13 不同运动时间 T_r 对应的圆弧曲线

特别指出的是，由于目标轨迹规划算法中控制电压正比于反馈偏差，而伺服电机转速又正比于控制电压，即伺服电机转速正比于反馈偏差。偏差越大速度越大，偏差越小速度越小，因此，目标轨迹规划算法适用于低速高精度应用中。

6.3 系统实现中的几个关键问题

6.3.1 RTX 环境下的数据采集

RTX 环境下编码器脉冲的采集是一个非常重要的问题，它是构成系统位置环反馈的关键环节。由于 RTX 是在 Windows NT 硬件抽象层的基础上扩展出自己的抽象层，可以直接对硬件进行操作，原来基于 Windows NT 的数据采集卡的驱动程序不能在 RTX 环境

下继续使用，有必要在 RTX 环境下实现对该卡的操作。数据采集卡的硬件已封装成集成芯片的形式，要获取卡上的接口信息是比较繁琐的，实际对数据采集卡的操作是通过修改该卡的 C 文件来实现的，因为其 C 文件内包含了卡的硬件信息。为了调用方便，我们保持该卡原来的函数形式，只是修改函数的具体实现。

通过对 C 文件的分析，我们发现函数实现调用了 Windows NT 的系统函数，而这些函数在 RTX 环境下是无法识别的，因此我们将这些函数转化成 RTX 自身的 API 函数。这些函数替换如下：

（1）_outp(wPortAddr,bOutputVal); 替换成：

RtWritePortUchar((PUCHAR)wPortAddr,bOutputVal);

（2）_outpw(wPortAddr,wOutputVal); 替换成：

RtWritePortUlong((PULONG)wPortAddr,wOutputVal);

（3）_inp(wPortAddr); 替换成：

RtReadPortUchar((PUCHAR)wPortAddr);

（4）_inpw(wPortAddr); 替换成：

RtReadPortUlong((PULONG)wPortAddr);

6.3.2 RTX 环境下的程序调试

RTX 的编译过程比较麻烦，使机器人运动调试期间更改程序非常繁琐，给控制系统的快速开发带来了很大不便。因此，有必要寻找一种加快程序开发的方法。通过对机器人控制系统总体结构的分析我们发现，系统的 Win32 环境只提供人机界面、机器人运动仿真和上层运动任务的传达，具体某项任务的实现没有必要用到 Win32 环境和通信层，故在底层运动调试的时候去掉 Win32 模块和底层通信层，这样生成的 RTSS 执行器可以直接在实时子系统成功运行；Win32 通信层和 RTSS 通信层在调试的时候也可以舍去人机界面和运动控制层；人机界面调试的时候可以舍去上下通信层和运动控制层；计算机仿真的时候又可以不考虑人机界面、通信层和运动控制层。这样做使得开发速度大大提高，同时程序调试更简单可行。可直接运行的 RTSS 执行器需要做以下设置：

（1）从菜单 Build 的 Set Active Project Configuration 中选择 RTController-Win32 RTSS Debug；

（2）在菜单 Project 的 setting 中的 Debug 页面中，Executable for debug session 项中写入 c:\program files\vci\rtx\bin\rtssrun.exe，然后在 Program arguments 项中写入 RTSS 执行器的路径及执行器名。

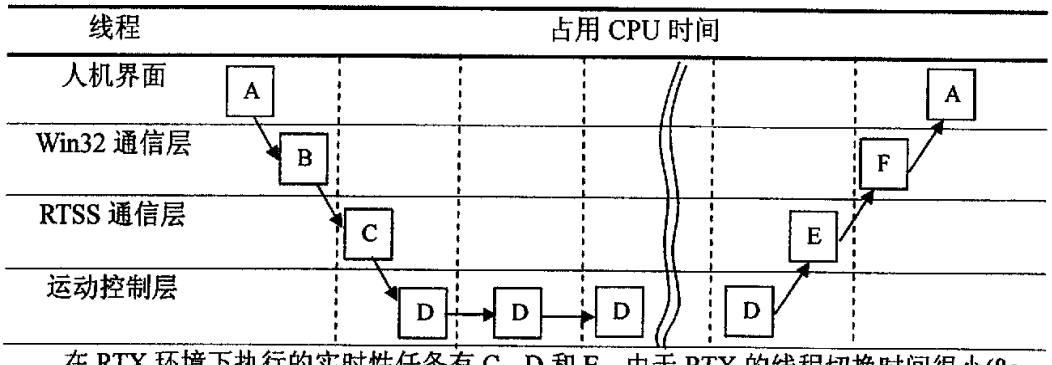
6.4 系统的性能分析

RTX 作为一种基于 Windows NT 的硬实时操作系统，无论在中断延时还是定时精

度上都显示出了卓越的特性。在一台普通配置的 x86 微机上，RTX 最大中断延迟时间不超过 15us，最大任务切换延迟小于 35us。这些实时性能参数与系统负载无关，只取决于计算机硬件。RTX 的定时器的分辨率为 1us，最小计时周期可以精确到 100us[28]。

不过机器人控制系统对实时性要求非常严格，各控制任务必须在一定的时间内完成，否则会对机器人的性能造成严重影响，甚至发生不可预知的事故。所以有必要对机器人的实时任务做出正确的规划，以保证实时性。在机器人实时系统的设计时必须考虑各任务对应线程的优先级，在基于优先级的线程调度算法下，重要任务可以实时地执行。机器人信息反馈线程对实时性要求最低，故将其优先级设成最小。伺服控制线程优先级最高，主要进行机器人各种运动的实时控制，如机器人状态监控、读取关节限位开关信号和关节码盘位置信号、各关节位置 PID 调节、通过 D/A 卡输出电机控制模拟量。Win32 线程比任何 RTSS 线程的优先级都低，其任务不要求实时完成。各线程占用 CPU 的时间见表 6-4，表中 A 为用户界面线程，用来发送控制指令；B 为 Win32 指令发送线程；C 为 RTSS 指令接收线程；D 为伺服控制线程；E 为 RTSS 信息反馈线程；F 为 Win32 信息接收线程。我们对这些线程所需要的时间周期进行定义，并分别用 T_A、T_B、T_C、T_D、T_E、T_F 和来表示。它们彼此之间的数量关系是 $T_A>T_F>T_B>T_E>T_C>T_D$，根据速率单调算法的原理，时间周期越短的任务，其发生频率越高，所以其实时性的要求也就越高，相应的，也应该分配给其更高的优先级。

表 6-4 机器人控制系统任务时序



在 RTX 环境下执行的实时性任务有 C、D 和 E，由于 RTX 的线程切换时间很小(8～20μs 左右)，在各任务之间相互切换时占用的 CPU 时间很少，故各任务都可以得到所需的执行时间，保证控制系统稳定运行。图 6-14 是利用 RTX 测试软件进行任务切换时间的测试，结果表明，切换最小时间为 8us，最大时间为 13us，平均切换时间约为 8us。

由于伺服控制线程 D 目标是完成机器人各种运动的实时控制，有必要分析其实时性能。我们运用 RTX 获取当前时钟值函数 RtGetClockTime()，精确获取 RTX 实时硬件扩展层提供并有 1us 的精度的时钟，计算圆弧轨迹运动（T_r =13.688s）目标规划算法的循环周期，并将循环周期存入文件，运用 MATLAB 工具进行分析，结果如图 6-15 所示。从图中可以看出，目标规划算法的循环周期维持在 1ms 左右，最大周期为 1004us，最小周期 996us，即最大偏差 4us，相对 1ms 偏离 0.4%，已达到很高的精度，满足实时性要

求。

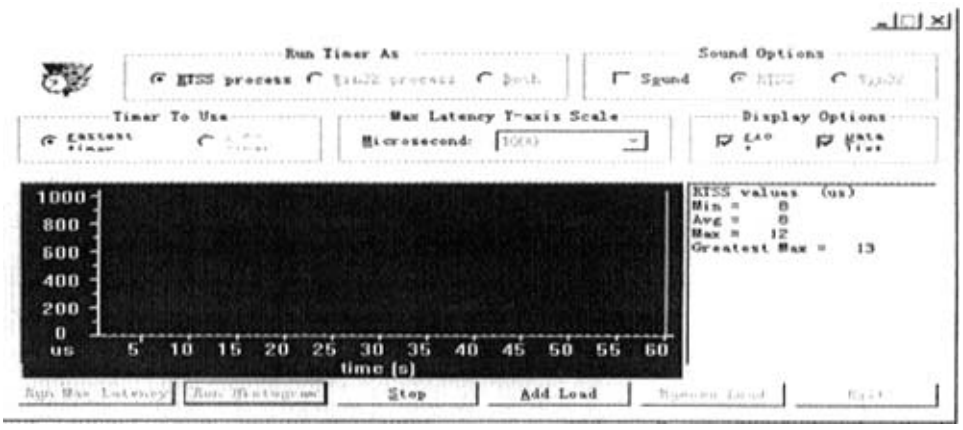


图 6-14 RTX 任务切换时间测试

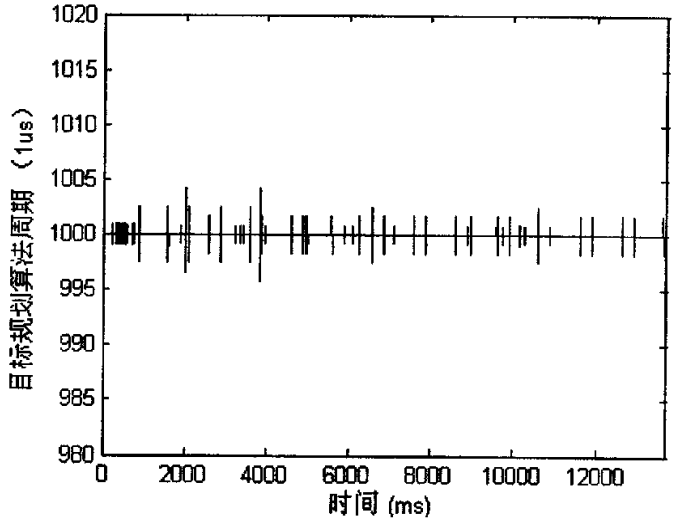


图 6-15 目标轨迹规划算法周期

6.5 本章小结

本章进行了机器人伺服系统的建模及伺服控制算法的具体介绍，并进行了机器人手臂单关节验证和双关节运动应用。当然，在系统实现中碰到了许多难题，其中重要问题的解决在 6.3 节做了介绍。为了保证系统的实时性，我们从 RTX 系统实时性、机器人控制系统调度和机器人控制算法实时性几个方面做了论证。

第七章　结论与展望

7.1　全文总结

随着工业化进程的推进，机器人作为一门新兴的学科逐步在越来越多的领域内获得人们的青睐并取得惊人的业绩。自 20 世纪 80 年代中期开始，我国机器人的发展已有 20 多年的历史，在这期间，我国在机器人技术方面取得了许多重要的成果，并缩短了与国际先进水平的差距，但是至今仍未形成大规模的机器人产业。大力展开机器人教学及研究，将十分有利于我国机器人产业的发展。本文使用 Windows NT+RTX 实时平台展开对双臂教学机器人伺服系统的研究，论文着重论述了如下几点内容：

（1）机器人运动分析及仿真

介绍了机器人的结构特点，进行了运动学正解和逆解，在机构简化的基础上给出了机器人的运动模型，然后在运动学正解和逆解的基础上运用 Visual C++6.0 完成了运动区域仿真和各种运动轨迹仿真。由于机器人的底层控制软件也用 VC 开发，可将其和仿真软件相结合，实现对机器人运动的控制；同时可以灵活的从仿真中获取有关数据。

（2）机器人软件总体结构的确定

阐述了机器人单处理器结构开放式控制器的思想、优点，进行了机器人实时系统的选择，说明其能够满足运动控制实时性的原理和机制，在此基础上给出了伺服系统的总体结构。

（3）系统的软件设计

通过对 Windows NT+RTX 实时平台的分析，提炼出了一种单处理器体系结构下分层结构的机器人控制系统设计方法，以适应教学机器人开放性好、成本低的各种需要。控制系统由人机界面、Win32 通信层、RTSS 通信层、运动控制层组成。人机界面和 Win32 通信层在 Win32 非实时子系统内实现，RTSS 通信层和运动控制层在 RTSS 实时子系统内实现，依靠共享内存等 IPC 机制来保证非实时系统和实时系统间的通信。

（4）伺服控制研究

进行了机器人手臂运动建模，在此基础上探讨了伺服系统的目标轨迹规划控制算法，并给出了对应算法的运行结果和性能分析。算法结果表明，目标轨迹规划算法在中低速时，手臂末端的位置误差较小（手臂末端线速度等于 87 mm/s 时对应手臂末端位置误差 0.619mm，手臂末端速度为 22.9514mm/s 时对应手臂末端位置误差 8.7462e-004mm），适合中高精度应用的各种控制需要。接着对控制系统实现过程中的关键问题提出了解决方法。然后对系统进行了性能分析，以保证控制系统的可靠性、稳定性、精确性。

7.2 前景与展望

基于单处理器的开放式机器人控制系统是一款实时、高效、可行的机器人控制系统，能够完成以往需要主从两级处理器来进行控制的实时任务，这样既简化了控制系统的结构，也提高了实时任务和非实时任务之间的通信速度。此外，主操作系统采用 Windows NT，由于 Windows NT 是当今最流行的操作系统，十分有利于该控制系统的推广和应用。

在微处理器性能的不断提高、硬件成本的不断下降的形势下，基于单处理器体系结构的开放式机器人控制系统将成为一种趋势，会有越来越多的人研究和应用它。

当然，系统的实现中会出现许多难题。运用目标轨迹算法进行机器人控制时，手臂末端速度在 100mm/s 以上开始运动时会出现振动，反映在运动轨迹上为纹波，需要应用其他控制方法来改善性能。

需要特别指出，双臂教学 SCARA 机器人有待进一步研究，一套高效双臂教学机器人控制系统的开发并不是一次研究便能够完成的，它的完善需要许许多多研究人员的共同努力。尽管机器人研究的非常艰辛，我仍坚信双臂教学 SCARA 机器人的研究会做的更好，因为中国的机器人产业正朝气蓬勃的发展着！

参 考 文 献

[1] 张铁，谢存禧. 机器人学[M]. 广州：华南理工大学出版社，2001

[2] 国家 863 计划智能机器人专家组. 机器人博览[M]. 北京：中国科学技术出版社，2001.

[3] 孙永发. 用微型电子和打乒乓球机器人进行教学的经验[J]. 机器人技术与应用，1995(4)：2-5.

[4] Dodds G. Robotics in Japan:from east to west and outer space to inner space[J]. Comput. and Contr. Eng.J, 1992,3(3): 143-148.

[5] 崔世钢等. 机器人与教育改革[J]. 北京：机器人技术与应用，2000(4)：16-17.

[6] 周学才，李卫平，李强. 开放式机器人通用控制系统[J]. 机器人，1998，20(1).

[7] G.Pritshow. Open System Controllers – A Challenge for the Future of the Machine Tool Industry. Annals of the CIRP Vol.42/1/1993: 449-452

[8] Anony mous.European research will result in open CNC architecture. Machinery and Production Engineering,1995(2):7-8

[9] Frederick Open-architecture Controllers[M]. IEEE Spectrum,1997(6):60-64.

[10] 张广立，谈世哲，杨汝清. 基于 Windows NT 的开放式机器人控制系统[J]. 机器人，2002，24(5):443-446.

[11] "Solutions for control" in Catalogue. Northville, MI: dSpace Inc.,1997.

[12] 王越超，王旭，谈大龙. 精密 1 号装配机器人控制器的设计与实现. 机器人，1997(1)

[13] D. Lim and H. Seraji, "Configuration control of a mobile dexterous robot: Real-time implementation and experimentation," Int. J. Robot.Res., vol. 16, no. 5, pp. 601–618, 1997.

[14] 牧野洋，谢存禧，郑时雄. 空间机构及机器人机构学[M]. 北京：机械工业出版社，1987.

[15] 梅志千，谈世哲，张广立，等. 一种新的检测仪传动机构的设计[J]. 机械科学与技术，2002，21(4): 588－590.

[16] 柳洪义，宋伟刚. 机器人技术基础[M]. 北京：冶金工业出版社，2002.

[17] 徐益，颜文俊，诸静. 机器人三维图形仿真的实现[J]. 计算机仿真，2003，20（7）：72-75.

[18] 申永胜. 机械原理教程[M]. 北京：清华大学出版社，2002.

[19] 何炎祥. 操作系统原理[M]. 北京：科学出版社，2004.

[20] 范永，谭民. 机器人控制器现状和展望[J]. 机器人，1999，21(1).

[21] 赵龙，胡宁. 对 UNIX 实时扩充的一点研究[J]. 计算机工程与科学，1998，20(4).

[22] 邹思轶. 嵌入式 LINUX 设计与应用[M]. 北京: 清华大学出版社，2002.

[23] （美）Stu Sjouwerman,Ed Tittel, Windows NT 超级工具[M]. 北京-中国电力出版社，

2001.

[24] (美)卡塔尼尔. Windows NT4 使用详解[M]. 北京：机械工业出版社，1998.

[25] (美)里克特. Windows NT 高级编程技术[M]. 北京: 清华大学出版社，1994.

[26] (美)昆特(Gunter, D.)，Windows NT 和 UNIX 集成指南[M]. 北京：机械工业出版社，1998.

[27] (美)迈门(McMains, J.),Windows NT 备份与恢复[M]. 北京：机械工业出版社，1998

[28] VenturCom Company, RTX 4.3 User Guide, 1999.

[29] VenturCom Company, RTX 4.3 Reference Guide, 1999.

[30] (美)贝茨. 实用 Visual C++6.0 教程[M]. 北京：清华大学出版社，2000.

[31] 黄维通，姚瑞霞. Visual C++程序设计教程[M]. 北京：机械工业出版社，2001.

[32] 陈永春. MATLAB M 语言高级编程[M]. 北京：清华大学出版社，2004.

[33] 苏金明，阮沈勇. MATLAB 6.1 实用指南[M]. 北京:电子工业出版社，2002.

[34] 梅志千. 机电伺服系统中的补偿技术研究[D]. 上海：上海交通大学机械与动力学院，2003.10.

[35] 胡寿松. 自动控制原理[M]. 第 4 版. 北京：科学出版社，2001.

[36] 蔡自兴. 机器人学[M]. 北京: 清华大学出版社，2000

[37] 谈世哲. 工业机器人控制器开放性、实时性分析方法以及单处理器模式下的实现[D]. 上海:上海交通大学机械与动力工程学院，2002.

致　谢

本课题的研究工作是在我的导师梅志千教授的直接指导下完成的。首先十分感谢梅老师对我理论上的指导和生活上的关心！在读研的时光中，梅老师的一言一行都给予了我莫大的影响，让我终身受益。

感谢上海交通大学硕士研究生王剑辉和张广立老师的热心帮助，他们曾耐心地解答了研究期间遇到的诸多疑点，他们刻苦钻研、精益求精的精神永远是我学习的榜样！

感谢我的父母、家人，正是他们多年来的无私奉献、鼓励和支持，才能使本人顺利完成了学业！对还在农村奋斗的家人，我要说声：辛苦了！

最后向所有关心和帮助过本人的老师和同学们表示最诚挚的谢意！

作者：赵小英

2006 年 1 月