文章编号: 1002-0446(2001)04-0356-07

基于网络和 Linux 的机器人仿真和监控系统*

陈一民　张　涛　薛广涛

(上海大学 计算机工程与科学学院　上海　200072)

摘　要: 本文介绍了一种新的基于网络的机器人仿真、监控系统, 该系统运行于 Linux 操作系统的 PC 机上, 采用 OpenGL 图形库进行开发, 能够实时模拟机器人的运动, 并且通过网络与机器人控制器进行通讯, 能够实时接收机器人控制器发来的状态数据, 将它们动态地以三维模拟方式显示或者将其存储起来以备将来分析, 用户能够动态地监视机器人的运动状态, 在必要时对机器人的动作进行控制.

关键词: 机器人仿真; OpenGL; Linux; 网络通信; 机器人建模

中图分类号:　TP24　　文献标识码:　B

A ROBOT SIMULATION AND MONITORING SYSTEM BASED ON NETWORK AND LINUX

CHEN Yi-ming　ZHANG Tao　XUE Guang-tao

(*College of Computer Engineering and Science, Shanghai University, Shanghai,* 200072)

Abstract: A new robot simulation and monitoring system has been developed in our unit. Being run on Pentium PCs with Linux operating system and being developed with OpenGL, this system has the capacity of simulating the motion of an industrial robot in realtime. Connected with the robot controller via network, the system can receive the status data of the robot and display them in 3D-simulation mode and/or store them for future analysis. Users can monitor dynamically the behavior of the robot, and can control directly the action of the robot if necessary.

Keywords: robot simulation; OpenGL; Linux; network communication; robot modeling

1　引言(Introduction)

随着计算机网络技术的迅猛发展, 对机器人通用性、开放性及编程系统的要求越来越高. 为适应上述要求, 我们建立了一种新的基于网络和 Linux 操作系统的机器人仿真系统, 目前 Linux 操作系统在全世界已经占据 17% 的市场份额, 形成了与 UNIX、Windows NT 三足鼎立的局面, 它遵循"自由"、开放源代码为宗旨. 该机器人仿真系统采用 OpenGL (Open Graphics Library) 图形库进行开发, 运行于奔腾级 PC 机上, 能够实时模拟机器人的运动, 并且通过网络与机器人控制器相连, 仿真系统能够实时接收机器人控制器发来的状态数据, 动态地以三维模拟方式显示或者将数据存储起来以备将来分析, 因此用户能够通过网络动态地监视机器人的运动状态, 并能对机器人的动作进行远程控制.

由于引入了仿真技术, 机器人作业示教编程系统具有如下主要等特点: 一是不用让机器人动作就能预先对其运动情况进行检查; 二是适用范围广, 从技术上讲, 只要能够建立起几何模型和运动模型就可以对各种机器人进行编程和仿真; 三是便于和 CAD/CAM 系统集成; 四是可以进行碰撞和干涉检验, 很好地保证了编程者和机器人的安全.

机器人的性能在很大程度上是由控制器决定的. 传统机器人控制器多以实现本单元的内部控制为主要目的, 缺少对外通信的能力, 缺乏与外界进行全面信息交换的能力, 缺乏与外界设备间进行协同工作的能力, 也缺少对外界环境的适应能力. 我们所研制了具有网络接口的机器人通用控制器, 正是通

* 基金项目: 国家教委《高等学校骨干教师资助计划》项目.
收稿日期: 2000-11-20

过引入了网络通信技术使系统的功能得到了加强, 也使机器人仿真系统的开发成为可能.

本文将对机器人仿真系统开发中一些关键技术: 如 Linux 平台上 OpenGL 三维显示技术; 机器人几何建模技术; 运动学建模技术和 Linux 平台上网络编程技术等进行研究说明.

2　Linux 下 OpenGL 三维显示技术 (OpenGL 3D display technology under linux operating system)

OpenGL 是由 SGI 公司开发的可独立于窗口系统, 操作系统和硬件环境的图形开发环境三维图形库. 由于 OpenGL 具有着跨平台性、简便性、高效、功能完善等特性, 已经成为了三维图形制作方法中事实上的工业标准, OpenGL 包括了大约 120 多条不同的命令, 用来定义 3D 物体和交互式 3D 应用的各种操作. 基于 OpenGL 开发的大量应用系统已广泛地应用于科学计算可视化、CAD/CAM、图像处理、地理信息系统、虚拟现实等领域.

虽然 Linux 下有 OpenGL 的商业版本, 但 Bran Paul 开发一种公开的、高质量的类 OpenGL 的三维图形开发库—Mesa. 由于没有获得 SGI 公司的许可, Mesa 还不能说是 OpenGL 的一种实现, 但它确实是一种 Linux 环境下非常有效的 OpenGL 编程工具. 从 Masa 3.0 版本起, 实现了 OpenGL1.2API 规范, 绝大多数的 OpenGL 应用程序不需要修改就能使用 Mesa 三维图形开发库运行. 我们开发本系统使用的是 Mesa 3.1 版本.

GTK(Gimp Tool Kit)用于 Linux 及 Unix 之上的 X-Windows 应用程序开发中, 用 GTK 可以开发出友好的图形交互界面, GTK 也是由事件驱动的开发工具. GTK 有称为 GtkWidget 的 C 数据结构的窗口部件, 具有面向对象风格. GtkGLArea 是 GTK 下实现 OpenGL 的构件(Widget), GtkGLArea 是包装了 GLX(OpenGL 针对 X-Windows 系统的扩展)函数的 gdkgl, 它本身由 GtkDrawinigArea 构件派生, 使用 GtkGLArea 就能在 GTK 下像使用其他的构件一样, 方便的调用 OpenGL/Mesa 函数库, 实现三维图形编程.

下面简单介绍一下应用 GTK 和 GtkGLArea 实现三维图形编程的过程:

1. 初始化: 调用 gtk_init() 进行 GTK 的初始化.

2. 建立顶层窗口: 调用 gtk_window_new (GTK_WINDOW_TOPLEVEL) 生成程序的主窗口, 作为其他构件的父构件.

3. 建立构件对象: 创建 GtkGLArea 对象, glarea = gtk_gl_area_new(attrlist). 创建程序中需要的其他对象, 如菜单、工具条、按钮等.

4. 建立信号和回调函数: 登记 GtkGLArea 对象感兴趣的事件, 如下对 GtkGLArea 对象 glarea 登记了重绘和鼠标按下、释放事件.

```
gtk_widget_set_events (GTK_WIDGET (glarea),
GDK_EXPOSURE_MASK|
GDK_BUTTON_PRESS_MASK|
GDK_BUTTON_RELEASE_MASK);
```

然后向 GTK 登记事件处理函数,

```
gtk_signal_connect (GTK_OBJECT(glarea), "realize",
GTK_SIGNAL_FUNC (glarea_init), NULL);//登记 glarea 初始化信号的处理函数是 glarea_init
gtk_signal_connect (GTK_OBJECT(glarea), "expose_event",
GTK_SIGNAL_FUNC (glarea_draw), NULL);//登记 glarea 重绘信号的处理函数是 glarea_draw
gtk_signal_connect (GTK_OBJECT(glarea), "configure_event",
GTK_SIGNAL_FUNC (glarea_reshape), NULL);//glarea 大小变化的处理函数是 glarea_reshape
```

再回调函数对事件进行处理, 例如 GtkGLArea 对象的场景绘制函数

```
glarea_draw (GtkWidget * widget, GdkEventExpose* event)
{
if (event->count > 0) return(TRUE);//只绘制最后一次的重绘事件
if (gtk_gl_area_make_current(GTK_GL_AREA(widget)));//在对 GtkGLArea 进行绘制前必须
调用 gtk_gl_area_make_current
{
glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);//在绘制前清除色彩和深
```

度缓存区

……;//在这里加入具体的绘制的代码

gtk_gl_area_swapbuffers (GTK_GL_AREA (widget));//使用双缓存, 交换前后缓存

}

}

5. 实现构件对象: 设置构件特性, 放置构件到各自的容器中并使其可见.

6. 进入消息循环: 调用 gtk_main(). 在消息循环中, 对鼠标、键盘以及系统消息进行响应, 当然也包括 OpenGL 的场景初始化状态设置和场景绘制.

以上编程的主要工作是构件对象的建立和场景绘制部分, 前者是用户图形界面设计的内容, 使用 GTK 提供的各种编程接口来实现, 后者是进行图形处理的关键部分, 通过 GtkGLArea 构件使用 OpenGL/Mesa 提供的各种库函数来实现.

3 机器人建模技术 (Robot modeling technology)

机器人的运动是由关节的运动所引起的, 而关节的位置是由连杆的长度和排列方式所决定, 与连杆的具体形状并无关系. 因此, 我们在研究机器人的运动机理时, 可对机器人进行某种抽象, 将连杆抽象为一条直线, 而将关节抽象为一点. 但是, 为了真实地仿真机器人的动作, 在进行三维显示时又必须使每个杆件尽可能与实物完全一致, 这样才能达到仿真的目的, 这一过程称为几何建模. 对机器人的建模实际上分为机器人运动学建模和机器人几何建模两部分.

3.1 机器人几何建模

OpenGL 虽然提供了很强的图形功能, 但是三维建模功能则相对薄弱, 仅在辅助库中提供了一些三维形体绘制函数, 仅仅能够完成对如球、立方体、多面体等简单形体的绘制. 除此以外, 提供了一些基本图元, 如点、线、多边形等, 要想直接用这些基本图元来完成复杂形体的建模工作, 是一件复杂的工作.

现有的 CAD 如 AutoCAD、UG II、Solidwork 等系统虽具有很强的三维绘图能力, 但利用 CAD 软件完成对机器人各个杆件的建模后, 从标准 CAD 数据文件(如 DXF、IGES 等)提取出几何形体数据用于 OpenGL 的绘图是关键.

VRML(Virtual Reality Modeling Language)语言是 Web 上广泛使用的三维图形标准, 目前许多 CAD 软件都具备了读入和输出这种格式文件的功能. 由于 3D Studio Max 和 AutoCAD 为同一家公司 - Autodesk 所出品, 故在 3D Max 中能够很好地实现 DXF 文件到 VRML 等文件格式的转换, 从而间接解决 DXF 文件的读取问题.

VRML 是 ASCII 码文本方式的文件, 便于理解, 通常将一个复杂形体表示成一个个小的三角形, 数据格式通常都是由两组数据组成, 一组为结点的三维坐标, 另一组用于表示节点之间拓扑关系.

在 VRML 中用 Shape 节点存放需要渲染的物体, 它的格式为:

```
Shape {
exposedField SFNode appearance NULL
exposedField SFNode geometry NULL
}
```

其中 appearance 域定义了 Appearance 节点, 在 Appearance 节点中存放 geometry 中使用的可视属性(例如材质和纹理); 而 geometry 域定义了一个 Geometry 节点, Geometry 节点由 Box、Cone、Cylinder、ElevationGrid、Extrusion、IndexedFaceSet、IndexedLineSet、PointSet、Sphere 和 Text 节点组成, 分别存放着三维的点、线、面、文字、实体.

通常使用 IndexedFaceSet 节点存放三维形体, IndexedFaceSet 节点中比较重要的是 coord 、normal 、coordIndex 和 normalIndex 域. coord 域定义了 Coordinate 节点存放顶点的坐标; normal 域定义了 normal 节点存放法向量列表; coordIndex 域定义了多边形面的构成信息, 数字表示顶点号, 而 - 1 则表示该多边形面结束, 下面的数据为一个新的多边形的顶点信息; normalIndex 域定义了法向量的索引信息, 实际上即为法向量列表中的位置.

下面以一个简单的 VRML 文件为例进行说明, 它所表示的 3D 图形如图 1 所示.

```
# VRML V2.0 utf8
Shape {
  appearance Appearance {
    material Material {
      diffuseColor 0.7 0.7 0.7
      transparency 0
        }
  }
  geometry DEF 1-FACES IndexedFaceSet {
    ccw TRUE
```

solid TRUE

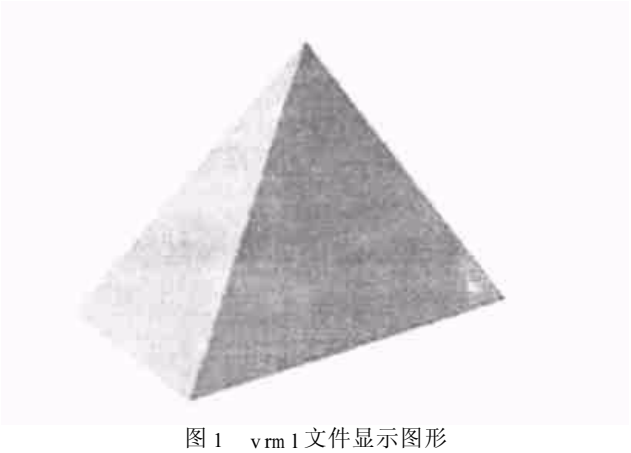

图 1　vrml 文件显示图形

Fig. 1　View of a vrml file

```
coord DEF 1-COORD Coordinate { point [
        0 30.8 0, - 25.08 - 6.161 13.86,
        25.08 - 6.161 13.86, 25.08 - 6.161 - 13.
86, - 25.08 - 6.161 - 13.86, 0 - 6.161 0]
        }
        coordIndex [0, 1, 2, - 1, 0, 2, 3, - 1, 0,
3, 4, - 1, 0, 4, 1, - 1,
          1, 5, 2, - 1, 2, 5, 3, - 1, 3, 5, 4, - 1,
4, 5, 1, - 1]
        }
    }
```

要从中提取出 3D 物体的数据就要编写关于 VRML 文件的语法分析器(Parser), 这是一个繁但不难的过程. 以上面的 coordIndex 域为例: 首先读到“coordIndex”保留字, 知道进入 coordIndex 域; 接着应该读到域分界符‘[’; 在没有读到域分解符‘]’之前, 所读取的应该是多边形面的构成信息: 读取数字, 直到遇到分界符‘,’(第一次读到的就是顶点 0, 第二次是顶点 1...), 将它保存到多边形面的队列中; 如果读到域分界符‘]’则 coordIndex 域的分析结束. 按照这样的方法就可以一步步提取出 3D 物体顶点和面的数据, 再通过 OpenGL 根据读取的顶点数据和面信息建立起各个多边形面元, 并将其存储到显示列表中, 在需要时予以调用.

3.2　机器人运动学建模

机器人的运动学分析(kinematic analysis)是对杆件、传感器等机器人的各个部件和作业环境内的对象等设置坐标系并分析这些坐标系之间的位置(position)和姿态(orientation)的关系. 杆件坐标系的选择有多种方法, 但最常用的是 Denavit-Hartenberg 方法(简称 D-H 法).

用 D-H 法对机器人进行运动学分析时先需对组成机器人的各杆件和关节进行编号. 将固定于基座的杆件设为 0, 然后从基座朝着末端执行器依次递增顺序编号; 对于关节也一样, 只要从底座开始朝着执行器方向依次对关节递增顺序编号即可. 下一步要进行的是对每个连杆建立连杆坐标系, 并确定相邻连杆间的关系. 图 2 和图 3 分别表示转动关节和平移关节的杆件坐标系设定的情况. 对于相邻的两个连杆 i 和 $i-1$ 及与之相连接的关节 $i-1$、i 和 $i+1$, 为了在连杆 i 建立连杆坐标系, 可将关节 $i+1$ 的旋转或平移轴定义为 z_i 轴, 并取旋转或平移的正方向为 z_i 轴的正方向; 取 z_{i-1} 轴与 z_i 轴的公共法线与 z_i 轴的交点为原点; 将 z_{i-1} 轴和 z_i 轴的公共法线有 z_{i-1} 轴向 z_i 轴方向延长, 取其延长线方向为 x_i 轴; 根据 x_i 轴和 z_i 轴来确定 y_i 轴, 使其构成右手系. 终端杆件的坐标系与距它最近的一个杆件坐标系的原点一致.

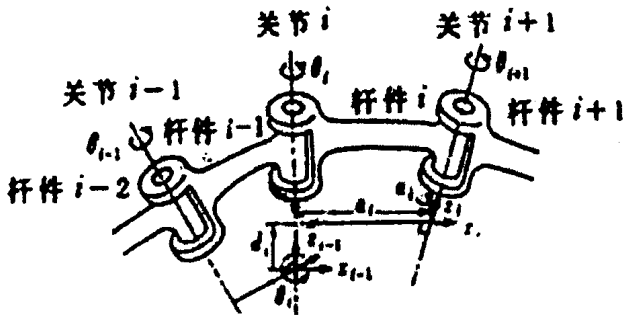

图 2　转动关节

Fig. 2　rotary joint

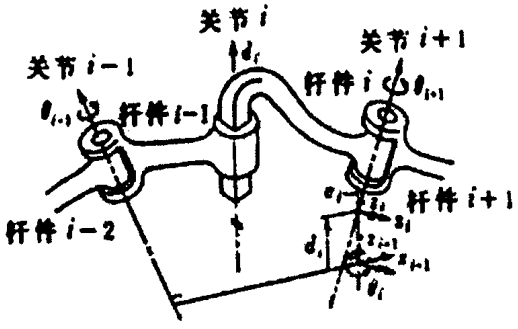

图 3　平移关节

Fig. 3　prismatic joint

对于任意一个连杆 i, 其位置与姿态可由 4 个参数来描述. 其中, 描述连杆 i 本身属性的参数有 2 个, 它们是连杆长度 ai 和连杆扭角 αi. 通过关节 i 相连的连杆 i 和连杆 $i-1$ 关系的两个参数分别为沿关节 i 轴线的两个公垂线的距离 di 与夹角 θi. 由连杆 $i-1$ 系到 i 系的变换为:

$$Ti = \mathrm{Rot}(z, \theta i)\,\mathrm{Trans}(0, 0, di)\,\mathrm{Trans}(ai, 0, 0)\,\mathrm{Rot}(x, \alpha i)$$

与此类似可以递推完成各连杆坐标系的变换. 假设 Tpq 表示连杆 p 到连杆 q 的变换, 则连杆 0(底座) 到连杆 q 的变换可表示为 $T0q$, 显然下列递推式成立:

$$T01 = T1$$

$$T02 = T01T2 = T1T2$$

$$T03 = T02T3 = T1T2T3$$

……

$$T0n = T0n-1Tn = T1T2T3\cdots\cdots Tn$$

若末端连杆为连杆 n, 则 $T0n$ 即为底座到末端执行器的变换式. 由以上递推式可以求出从底座到末端执行器间任意连杆的变换式. 只要每个杆件依照如前所述的方法选取坐标系来进行建模, 并给定每个杆件的 4 个参数, 就能够根据以上递推式, 在相应的位置正确绘制出各个连杆的姿态. 只要实时给出当前每个连杆的参数, 就能够完成机器人运动过程的实时模拟.

3.3 交互式运动学建模和几何建模

为满足通用化的要求, 适应为不同型号的机器人建立机器人运动学模型和几何模型, 我们开发了机器人模型定义程序. 通过该程序能够导入由 CAD 软件建立的机器人各个连杆的几何模型文件并显示, 由使用者来交互式地快速定义该机器人拓扑结构, 如由多少个连杆和关节构成, 及它们之间的相互关系, 几何参数(例如顶视图、正视图和左视图的视点的定义, 各个杆件的色彩定义等), 运动学参数(D-H 法中 a、α、d 参数, 特定机器人所需的限制条件)等. 机器人部件定义程序的界面如图 4 所示.

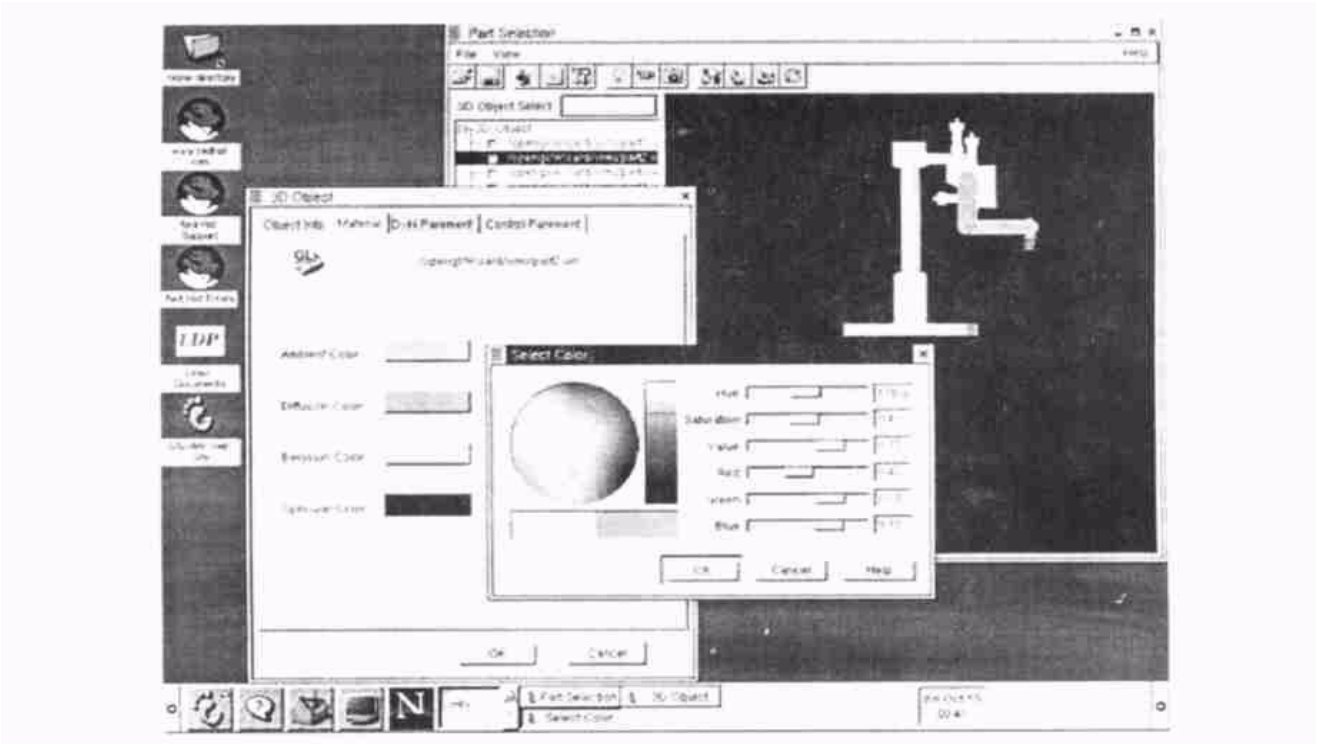

图 4 机器人模型定义程序

Fig. 4 Robot model definition program

可导入的几何模型文件包括 vrml 文件和 3DS 文件. 在完成了对机器人几何、运动学模型参数定义后, 可以将它们保存为该机器人的描述文件, 由图形仿真与监控系统调用.

4 在 Linux 下实现网络通信 (Network communication under linux operating system)

Linux 的网络实现支持 BSD 套接口, 支持完整

的 TCP/IP 协议. BSD 套接口是一种内部进程间的通信机制, 已成为网络编程的通用接口. 针对传输的数据需要连续数据流, 且需保证数据可靠的特点, 我们选择了流式 Socket 方式, 编程直接采用通用 Socket 的方案. Socket 的编程原理基于套接字的系统调用. 应用程序首先必须通过调用 Socket( ) 创建套接字. 然后调用 Bind( ) 将套接字地址与所创建的套接字联系起来. 通过 Connect( ) 和 Accept( ) 建立套接字的连接, Listen( ) 用于面向连接的服务器, 表示它愿意接受连接, Listen( ) 需在 Accept( ) 之前调用. 当一个连接建立后, 可以传输数据了, 常用的系统调用有 Send( ) 和 Recv( ). 最后调用 CloseSocket( ) 关闭套接字, 并释放所分配给套接字的资源.

由于 socket 进行传输的是字节流, 必须在其上定义自己的高层协议, 才能实际用于状态数据的传送. 我们定义了传输数据协议单元. 协议单元由一个单元头和若干个记录, 以及终结符组成. 记录的个数是可变的, 以适应不同场合的需要. 单元头和记录与记录之间由记录分隔符来进行分隔. 每条记录通常分为两个域, 域之间用域分隔符隔开. 第一个域表示记本记录的具体类型, 如是数据还是命令, 具体为什么命令等. 为简便起见, 直接采用一个 32 位整数来表示. 第二个域为该记录的具体数据, 数据间用域内分隔符隔开. 下面为一个协议单元的实例:

2# COMMAND* trace; home&ABSOLUTE. DEG* 100. 00; 30. 45; 92. 66; - 67. 10; - 25. 85@

“2”表示该单元中有两条记录, “#”为分隔符, 标志着第一条记录的起始. “COMMAND”表示该记录为命令; “trace, home”为命令的具体内容, 打开捕捉功能, 并执行回位操作. “*”为记录分隔符, 表示第一个记录已结束, 下面将开始第二个记录. ABSOLUTE. DEG 表示该记录为绝对角度坐标数据. 分隔符后面为具体的绝对坐标的具体实际, 各数据间用内部分隔符为“;”隔开. 整个单元由终结符“@”结束.

由于系统随时可能需要进行数据的发送和接收以得到需要的信息, 应用程序之间的数据传输很频繁且不定时. 而系统又要求数据的传输不能消耗过多的系统资源, 不能妨碍用户界面上的操作和显示. 这里可以使用多线程实现多任务并行, 达到系统的要求.

同时, 服务器方程序必须具备多机连接的功能, 既要处理监听套接口, 又要处理已连接套接口的通讯, 需要采用 I/O 复用, 即这样的功能: 如果一个或多个 I/O 条件满足(例如, 输入已准备好被读, 或者套接字可以承接更多的输出) 时, 我们就被通知. I/O 复用是由函数 select 和 poll 支持的, 使用函数 select 可以同时检查多个套接字是否就绪, 当有套接字就绪时, 函数 select 成功返回.

因此, 服务器方程序具体实现如下:

首先调用 socket( ), bind( ), listen( ) 函数, 建立套接字做好接收准备,

然后需要调用 ioctl(serversock, FIONBIO, &flag); //设置主套接字为非阻塞式套接字

FD. SET(serversock, &readfds); //打开主套接字的读监听位

接着调用 pthread. create(&tid, NULL, (void *) &SrvFunc, NULL); 创建后台工作线程 SrvFunc( ). 这时, 服务器方通讯的初始化完成, 返回避免由于通讯造成主程序阻塞而不能对机器人进行控制, 妨碍用户界面上的操作和显示的结果. 具体的通讯在后台工作线程 SrvFunc( ) 中进行, 它完成下述循环过程:

```
while(1)
{
select ( MAXCONNECTION, &rfds, &wfds, &efds, &TimeOut); //调用 select, 等待某个事件发生: 可能是新客户连接的建立; 也可能是数据、FIN 或 RST 的到达. 其中 MAXCONNECTION 表示最多可以连接的客户机的数量; 参数 TimeOut 定时器的值为 0, 则表示立即返回.
if ( FD. ISSET( serversock, &rfds)) //有新客户连接, 建立连接.
{
connfd= accept( serversock, &cliaddr, &client); //调用 accept, 建立与客户机的连接
for ( i = 0; i < MAXCONNECTION; i+ + ) //检查所有的连接
   if ( client[ i] < 0) {
client[ i]= connfd; // 保存与客户机通讯的套接字
break;
}
if ( i = = MAXCONNECTION)
err. quit( "too many clients"); //太多的连接客户
// 设置新的监听标志位
if ( connfd > maxfd)
```

```
maxfd = connfd;
if (i > maxi)
maxi = i; // client[ ]用户队列中最大的客户标号
}
for (i = 0; i < = maxi; i+ + ) // 检查所有连接的客户
{
if ( (sockfd = client[ i]) < 0)
continue; //连接的套接字已经关闭, 返回
if (FD. ISSET(sockfd, &rset)) //套接字可读
{
SrvReceive (sockfd, buf, 32); //调用函数SrvReceive 进行读取处理
}
}
}
```

与服务器程序相比, 客户机端程序略为简单, 这里就不赘述了.

5 应用实例 (Application examplex)

我们设计的系统分别应用于 Pt500 和 Pt600 机器人本体都取得了良好的性能. 如图 5, 图 6 分别是应用于 Pt500、Pt600 机器人的仿真监控界面. 图 8、图 9 分别是客户器端、服务器端的仿真监控程序同步运行的界面.

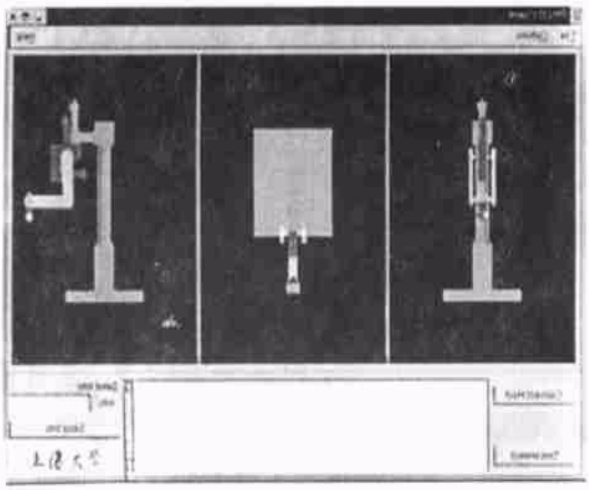

图 5 Pt500 机器人

Fig. 5 Pt500 robot

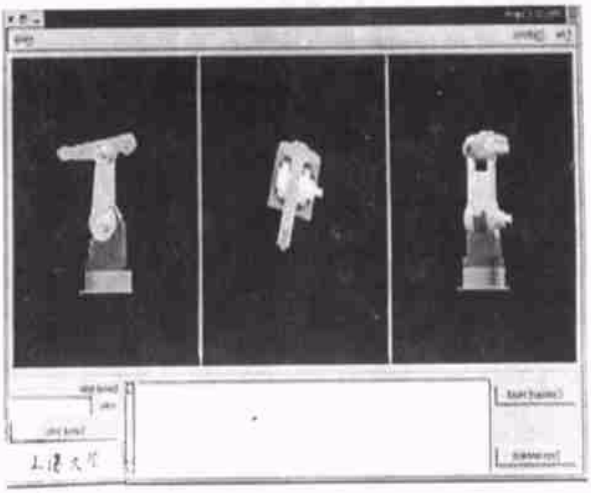

图 6 Pt600 机器人

Fig. 6 Pt600 robot

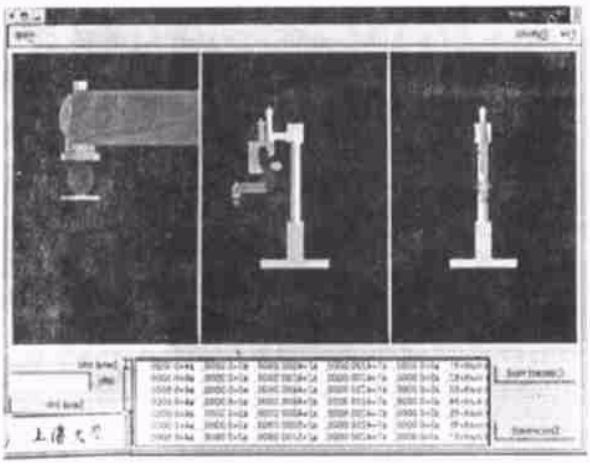

图 7 客户器端仿真监控程序界面

Fig. 7 Display window of simulating and monitoring program on client

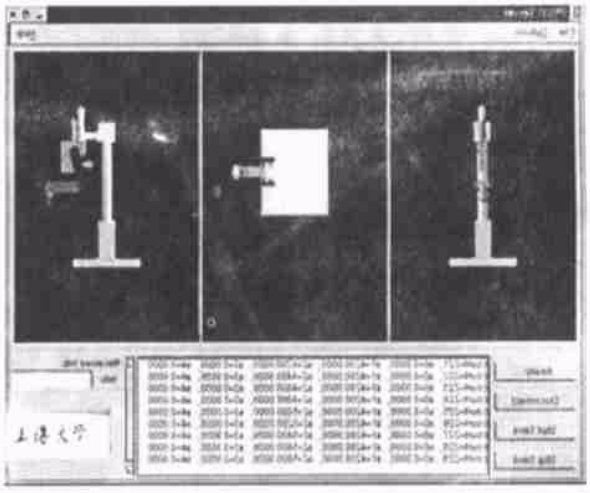

图 8 服务器端仿真监控程序界面

Fig. 8 Display window of simulating and monitoring program on server

(下转第 367 页)