DOI:10.19344/j.cnki.issn1671-5276.2020.05.041

工业机器人三维虚拟监控系统的设计与实现

孙恺廷[1],朱隽垚[2],于存贵[1],周成[1]

(1. 南京理工大学 机械工程学院,江苏 南京 210094;
2. 湖北江山重工有限责任公司 火箭炮研究院,湖北 襄阳 441057)

摘　要:针对目前对工业机器人缺乏有效的手段进行实时监控的问题,以 ABB-1400 六自由度工业机器人为研究对象,设计了一种能够反映工业机器人实时状态并具有良好人机交互能力和真实感的三维虚拟监控系统。系统基于 Unity3D 虚拟现实引擎,通过结合三维实体建模技术、模型渲染技术以及数据采集与传输技术实现了 B/S 架构下工业机器人三维虚拟监控系统。现场调试的结果表明,该系统能够实时再现工业机器人的真实状态。

关键词:工业机器人;监控系统;虚拟现实; ThingWorx 物联网平台

中图分类号:TP277; TH166　　**文献标志码**:B　　**文章编号**:1671-5276(2020)05-0154-03

Design and Implementation of 3D Virtual Monitoring System for Industrial Robot

SUN Kaiting[1], ZHU Juanyao[2], YU Cungui[1], ZHOU Cheng[1]

(1. School of Mechanical Engineering, Nanjing University of Science and Technology, Nanjing 210094, China;
2. Rocket Research Institute of Hubei Jiangshan Heavy Industry Co., Ltd., Xiangyang 441057, China)

Abstract: The current industrial robots lack the effective real-time monitoring. This paper takes ABB-1400 6-dof industrial robot as object of study and designs, a 3D virtual monitoring system, which can reflect the real-time status of the industrial robots and has man-machine interaction capability and real sensibility. In this virtual monitoring system, Unity3D engine is used with 3D solid modeling technology, model rendering technology, data acquisition and transmission technology to implement 3D virtual monitoring of the industrial robot under B/S architecture. The results of on-site commissioning show that the system can be used to monitor the real state of the industrial robot.

Keywords: Industrial robots; monitoring system; VR; ThingWorx

0　引言

随着工业机器人在工业领域的普及和推广,工业机器人将会越来越多地代替人完成一些高难度、高风险、工作环境恶劣的工作,在这种情况下如何实现对工业机器人的有效监控就显得非常重要了。近年来,随着计算机仿真技术的发展,基于实时数据驱动的工业机器人三维虚拟监控系统备受关注,国内外许多学者也对此进行了研究,但是普遍存在系统开发门槛高、开发效率低、系统可移植性差、监控方式单一等问题。

目前,可实现虚拟监控系统的方式大致可以分为 4 类[1-6]:

1) 利用三维动画软件实现

通过预先设计好三维模型的各种动作,如:转动、平移、缩放等,利用三维动画制作软件建模,生成控制对象各种角度、姿态对应的图片,并按一定的规则对其进行编号。在显示时,只需计算控制对象对应的图片编号并加以调用显示就可以达到实时仿真的目的。但是由于该方法只能在可知范围内对控制对象的简单动作进行动画演示,在控制对象动作复杂、运动趋势不确定的情况下采用三维动画制作软件的方式实现虚拟监控,往往会造成系统庞大、实时性差以及控制对象动作失真的情况。

2) 直接利用 OpenGL 实现

OpenGL(open graphics library,开放图形库)是一种适用于渲染 2D、3D 矢量图形的跨语言、跨平台的应用程序编程接口(API)。这个接口由近 350 个不同的函数调用组成,是一种独立于硬件设备和操作系统的图形与硬件的接口,开发人员可以利用它开发出功能强大的交互图形应用程序。由于 OpenGL 的核心 API 没有窗口系统、音频、打印、键盘/鼠标或其他输入设备的概念,这就意味着如果直接使用 OpenGL 开发虚拟监控系统需要自己完成窗口界面的设计、交互程序的开发以及模型对事件的响应等底层程序开发工作。这些程序开发工作要求开发者熟练掌握 C/C++和 Java 语言,这对非计算机专业的人员来说,直接使用 OpenGL 开发复杂的三维应用程序是比较困难的。

3) 利用 Web3D 技术实现

目前,Web3D 的实现有几十种可供选择的技术和解决方案,相关的软件有 30 多种,并且仍然有新的技术出现,使 Web3D 的渲染速度、图形质量、造型技术、交互性以及数据的压缩与优化等不断得到提高。典型的 Web3D 技术有: Cult3D、Java3D、Atmosphere、Direct3D、Viewpoint、X3D/VRML2.0(VRML1997)等。但是,由于缺乏统一的标

第一作者简介:孙恺廷(1993—),男,山东荣成人,硕士研究生,研究方向为先进制造理论及技术。

准,各种技术都是由不同的厂商自行开发的解决方案,使得每种技术使用的都是不同的格式和方法,而且几乎每个厂商开发的标准都需要自己的插件支持,这极大地限制了Web3D技术的应用。

4)基于虚拟现实技术的多软件协同实现

采用虚拟现实开发引擎如Unity 3D、IdeaVR,结合三维实体建模软件如Solidworks、Creo,模型渲染软件如Maya、3Ds Max协同开发虚拟现实系统。采用多软件协同开发的优势在于可以充分发挥各专业软件的性能优势,减少系统底层开发的工作量,降低了系统的开发门槛,缩短了系统的开发周期。

综上所述,本文选择第4种方式开发虚拟监控系统。具体来说,以ABB公司生产的六自由度串联工业机器人IRB1400本体和IRC5控制器组成的机器人控制系统为研究对象,通过Creo三维实体建模、3DS Max模型渲染、Unity3D虚拟场景构建以及TingWorx采集数据实现虚拟监控系统的成功上线。系统的开发可分为3个阶段:模型构建、实时数据采集传输以及基于实时数据的模型驱动。

1　模型构建

在工业机器人的三维虚拟监控系统开发中,首先要进行的就是三维模型的构建。建模是对现实对象或真实环境的模拟,逼真的模型是实现虚拟现实系统真实感和沉浸感的基础[7]。

本系统三维模型构建的具体步骤:

1)模型数据的采集

本系统以个性化印章生产线中的搬运机器人为原型,工业机器人的外观、几何参数来自于设备厂商,纹理信息来自实地拍摄的照片,其他场景、设备信息根据需求实地测量或者合理设计获得。

2)模型创建

采用Creo软件创建工业机器人三维实体模型。为提高三维虚拟监控系统的性能,在模型创建时就需要删除机械臂、控制柜内部等具体结构中一些不可见的、不必要的几何元素。

3)模型渲染

在3DS Max软件中,渲染创建好的三维实体模型,利用材质、纹理贴图模拟真实材料的质感,使模型具有真实材质纹理的感官效果。为方便后续在Unity3D中定义机械臂的动作,每一个运动单元都应是一个独立的物体,如图1所示。

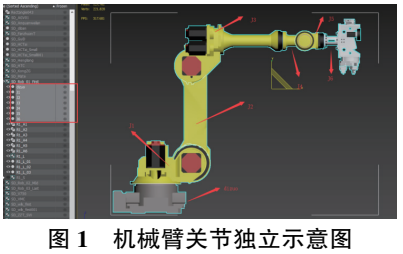


图1　机械臂关节独立示意图

4)场景搭建

将3DS Max软件渲染完成的模型以.FBX格式导入Unity3D中,然后将机械臂以及其他模型添加到场景的合适位置,调整模型各个属性的参数值,添加灯光效果,使模型具有光学上的真实感。场景模型如图2所示。

图2　系统场景模型

2　数据采集与存储

工业机器人三维虚拟监控系统是由真实的设备数据进行驱动的,因此需要将机械臂各关节的实时角度、末端的世界坐标以及末端执行器的状态信息采集、传输到数据库。具体实现过程可分为两步进行:使用ABB提供的PC SDK开发上位机程序读取需要监测的数据信息;基于Remoting协议采用C#语言编写应用程序将数据与ThingWorx进行对接。

2.1　数据采集

ABB提供的PC SDK允许系统集成商、第三方或个人用户为IRC5控制器添加自定义的应用程序。这种自定义的、独立的应用程序通过网络与机器人控制器进行通信,并可以作为独立的PC应用程序使用。本文通过该PC SDK开发的应用程序的功能主要包括两部分:扫描在线的ABB机器人;采集所有在线机器人的六关节轴角度以及末端的世界坐标和IO信息(末端执行器状态),数据采集结果如图3所示。程序运行结果表明有2台ABB机器人在线,这也是符合现场实际状态的。

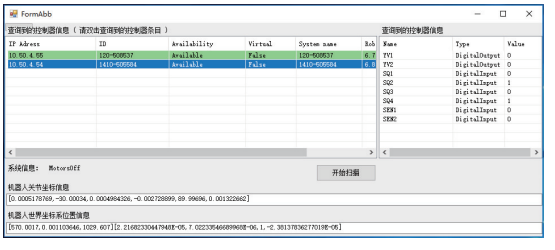


图3　工业机器人数据采集结果

2.2　数据传输与存储

ThingWorx提供了相应的数据访问接口,只需在开发环境中使用C#语言将数据与相应的接口进行绑定,就可以实现数据的上传;由于ThingWorx内部已经集成了PostgreSQL数据库,对于上传的数据会进行自动保存,这大大简化了开发过程。数据传输部分程序如图4所示。

3　基于实时数据的模型驱动

基于实时数据的模型驱动是实现虚拟监控系统的关键环节,其实现过程可分为以下4部分内容:建立合理的层次关系、从ThingWorx获取实时数据、对实时数据进行解析以及将数据与运动单元进行绑定。

```
static void Main(string[] args)
{
    ClassABB.Abb_Load();
    args = new String[3];
    args[0] = ConfigurationManager.AppSettings["PTC_Address"].ToString();    //访问地址
    args[1] = ConfigurationManager.AppSettings["Secret"].ToString();   //秘钥 Secret
    args[2] = "100";                                  //扫描速率
    if (args.Length < 3)
    {
        Console.WriteLine("Required arguments not found!");
        Console.WriteLine("URI AppKey ScanRate");
        Console.WriteLine("Example:");
        Console.WriteLine("MXROBOT ws ws://localhost:80/Thingworx/WS xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx 1000");
        return;
    }

    Console.WriteLine("连接地址:" + args[0]);
    Console.WriteLine("密钥 :" + args[1]);
    Console.WriteLine("扫描速率:" + args[2]);

    var config = new ClientConfigurator();

    config.Uri = args[0];

    // 如果出现断开或者初始无法建立连接，重新连接每15秒
    config.ReconnectInterval = 15;

    //设置连接方式
    var appKey = args[1];
    var claims = SecurityClaims.fromAppKey(appKey);
    config.Claims = claims;

    //设置客户端的名称
    //SteamSensorGateway
    config.Name = "MXROBOT";

    int scanRate = Int32.Parse(args[2]);

    // 构造函数 创建连接客户端的实例
    SteamSensorClient client = new SteamSensorClient(config);

    //        ConnectedThingClient client1 = new ConnectedThingClient(config);

    ConnectedThingClient.disableEncryption();//禁用证书验证
```

（a）建立通信连接

```
try
{
    // Set the property values
    base.setProperty("Robot1JointA1", ClassABB.rbt[o].Axi sCd[0]);
    base.setProperty("Robot1JointA2", classABB.rbt[o].Axiscd[1]);
    base.setProperty("Robot1JointA3", ClassAEB.rbto.Axi scd[2]);
    base.setProperty("Robot1JointA4", ClassABB.rbt[o].Axi scd[3]);
    base.setProperty("Robot1JointA5",  ClassABB.rbt[o].Ai scd[4]);
    base.setProperty("Robot1JointA6", ClassABB.rbt[O].Axi scd[5]);
    base.setProperty("Robot1YV1",ClassABB.rbtO].D0[1]);
    base.setProperty("Robot1Yrv2", clasABB.rbt[oi.D0[1]);
    base.setProperty("Robot1sQ1", cClassABB.rbt[oi.DI[0]);
    base.setProperty("Robot1sQ2", ClassABB.rbt[o.DI[1]);
    base.setProperty("Robot1sQ3", ClassABB.rbt[o.DI[2]);
    base.setProperty("Robot1SQ4", ClassABB.rbt[o.DI[3]);
    base.setProperty("Robot1SEH1", ClassABB.rbt[O].DI[4]);
    base.setProperty("Robot1SEN2", ClassABB.rbt[0].DI[5]);


    base.setProperty("Robot2JointA1", ClassABB.rbt[1].Axi sCd[o]);
    base.setProperty("Robot2JointA2", ClassABB.rbt[1].Axiscd[1:
    base.setProperty("Robot2JointA3", ClassAEB.rbt[1.Axi scd[2:
    base.setProperty("Robot2JointA4", ClassABB.rbt[1].Axi scd[3]);
    base.setProperty("Robot2JointA5", ClassABB.rbt[1].Axi scd[4]:
    base.setProperty("Robot2JointAB", ClassABB.rbt[1].AxisCd[5]);
    base.setProperty("Robot2YV1",ClassABB.rbt[1].D0[0]);
    base.setProperty("Robot2YV2", classABB.rbt[1].D0[1]);
    base.setProperty("Robot2M1",classABB.rbt[1].D0[2]);
    base.setProperty("Robot2SQ1", classABB.rbt[1].DI[0]);
    base.setProperty("Robot2sQ2", ClassABB.rbt[11].DI[1]);
    base.setProperty("Robot2SQ3", classABB.rbt[11].DI[2]);
    base.setProperty("Robot2SQ4", ClassABB.rbt[1].DI[3]):
}
atchl    (Excertion ex)
 {
        onsole.riteLine("communication root Failed : " + ex.Message);
}
```

（b）定义通信数据类型

图 4　数据传输部分程序

3.1　建立合理的层次关系

虚拟场景采用场景树进行管理,利用树结构来表现物体之间的层级关系,子节点是父节点的子物体,所有节点是根节点的子物体。工业机器人的层次关系如图 5 所示。

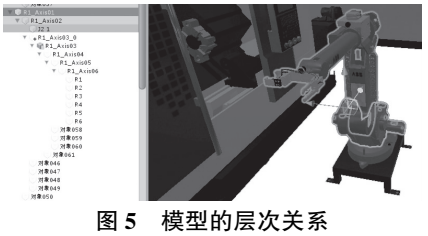


图 5　模型的层次关系

3.2　从数据库获取实时数据

ThingWorx 支持网页访问数据库,因此本文通过编写 C#程序访问特定网址的方式获取实时数据,并设定数据读取频率。update 函数可实现每秒 50 次的数据更新,但考虑实际应用需求和服务器处理能力,设定数据读取频率为 10。部分程序如图 6 所示。

3.3　对实时数据进行解析

因为数据库中包含车间所有设备的状态数据,如图 7 所示。因此在获取实时数据后,还需要对数据进行匹配和解析。部分程序如图 8 所示。

```
public class GetJson : MonoBehaviour
{


    private string JsonStr;
    private string GetServerUrl = "http://[redacted]Thingworx/Things/getDevicesData/Services/getJsonData?

    private bool IsAsk = false;
    private double ReaqIntT = 0;
     static JsonData tempdata;

    public bool Test_bool;

    public W_Updata w = new W_Updata();
    public R1_Updata r1_updata = new R1_Updata();
    public R2_Updata r2_updata = new R2_Updata();
    public S1_Updata s1_updata = new S1_Updata();
    public S2_Updata s2_updata = new S2_Updata();
    int a;
    public void ContrSend(bool temp)
    {
        IsAsk = temp;
        Debug.Log("请求状态" + IsAsk);
```

图 6　获取实时数据

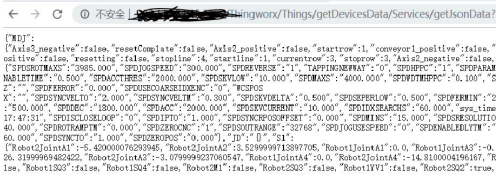


图 7　数据库中的实时数据

```
public void GetData(JsonData JsnD)
{
    Debug.Log(JsnD.ToJson());
    if (JsnD["S1"]["Robot1JointA1"] != null)
    {
        {
            Debug.Log((double )JsnD["s1"]["Robot1JointA1"]+"__—
            R1_angle[o] = (double)JsnD["s1"]["Robot1JointA1"];

        }
    }
    else
    {

        Debug.Log("数据初始化失败");

    }
    if (JsnD["S1"]["Robot1JointA2"]!= null)
    {
    R1_angle[1] = (double)JsnD["S1"]["Robot1JointA2"];

    }
    else
```

图 8　数据解析

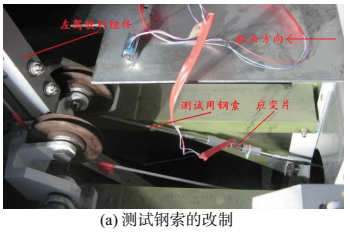
3.4　将实时数据与运动单元进行绑定

这里使用了 Unity3D 提供的 Dotween 插件,它提供了常用运动的 API 函数,可以快速对模型的运动进行定义。部分程序如图 9 所示。

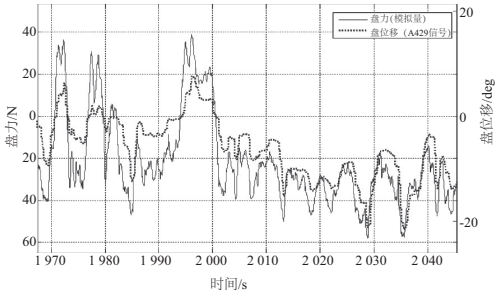
```
using UnityEngine;
using DG.Tweening;

public class R1_Move : MonoBehaviour {

    public Transform R1_Axis01;
    public Transform R1_Axis02;
    public Transform R1_Axis03;
    public Transform R1_Axis04;
    public Transform R1_Axis05;
    public Transform R1_Axis06;
    public float speed = 0.8f;

    public  void init()
    {
        R1_Axis01 = GameObject.Find("R1_Axis01").transform;
        R1_Axis02 = GameObject.Find("R1_Axis02").transform;
        R1_Axis03 = GameObject.Find("R1_Axis03").transform;
        R1_Axis04 = GameObject.Find("R1_Axis04").transform;
        R1_Axis05 = GameObject.Find("R1_Axis05").transform;
        R1_Axis06 = GameObject.Find("R1_Axis06").transform;
    }
    public void R1_Drive(params double[] R1_angle)
    {


        R1_Axis01.DOLocalRotate(new Vector3(0,0, -(float)R1_angle[0]),speed);
        R1_Axis02.DOLocalRotate(new Vector3((float)R1_angle[1], 0, 0), speed);
        R1_Axis03.DOLocalRotate(new Vector3((float)(R1_angle[2]- R1_angle[1]), 0, 0), speed);
        // Debug.Log((float)R1_angle[2]+"000000000000000000000000000000000000000000000000");
        R1_Axis04.DOLocalRotate(new Vector3(0, (float)R1_angle[3], 0), speed);
        R1_Axis05.DOLocalRotate(new Vector3((float)R1_angle[4], 0, 0), speed);
```

图 9　基于实时数据的模型驱动

(下转第 210 页)



(a) 测试钢索的改制



(b) 机上测试数据

图 7　钢索改制与机上测试数据

4　结语

飞机参数的测试不但要考虑参数测试的可能性,更重要的要考虑飞机各种使用状态下的安全性。本文通过对各种驾驶盘力测试方法的比较,创新性地提出了通过改造钢索进行测试的技术方案,并成功应用于飞机试验,解决了安全、改装空间、质量、测试精度等问题,并在此基础上形成了专利,创新性地为驾驶盘操纵力测量提供了有益的参考。

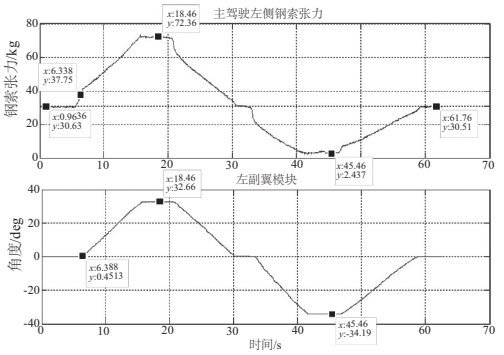


图 8　钢索张力试验室测试曲线

参考文献:

[1] 林丰俊. CRJ200 飞机飞行操纵系统设计分析[J]. 民用飞机设计与研究,2001(4):32-39.
[2] 郑培文,王俊元,张纪平,等. 电动操纵负荷系统机构设计与仿真建模[J]. 机械制造与自动化,2019,48(2):101-104.
[3] 李浩, 王海燕. 电传操纵系统在民用客机中的新发展[J]. 中国民航学院学报, 2000(6):8-11.
[4] 李良玉,张琦,张兆璨. 电传飞控系统曲线测量方法探究[J]. 科技风,2020(13):117-119.
[5] 马丽,崔海东,崔文宏.飞机人感系统浅析[J]. 无线互联科技,2014(7):49.
[6] 宋新贵, 张新国. 电传飞行控制系统[M]. 北京:国防工业出版社, 2003.
[7] 薛瀛, 郭建伟,田金强,等. 适用于钢索传动的电传飞机驾驶盘盘力测量方法研究[J]. 科技传播,2011(10):60-61.

收稿日期:2020－06－01

(上接第 156 页)

3.5　系统发布与调试

工业机器人三维虚拟监控系统最终是通过网页的形式进行发布的。本文通过示教器手动随机操作机器人的方式验证系统的监控效果,并通过调整 Dotween API 函数的参数,实现模型运动与真实运动的同步。现场调试过程如图 10 所示。

图 10　现场调试

4　结语

本文研究了一种基于多软件协同的方式实现工业机器人三维虚拟监控系统,提出一种基于 Creo、3DS Max、Unity3D、ThingWorx 软件相结合的工业机器人三维虚拟监控系统的快速开发方法。使用 SDK 编写自定义的数据采集程序实现工业机器人实时数据的上传,简化了系统的结构;采用虚拟现实的方式实现监控过程,其效果更加逼真,人机交互性更好;采用网页的形式进行发布,允许移动智能终端的访问,提高了监控的便捷性。由于本监控系统仅作为数字化车间三维虚拟监控系统的一部分内容,整体系统还未开发完毕,对工业机器人的监控只是作为系统可行性的验证,接下来将整合整个车间的生产过程,并在场景性能优化、人机交互、可视化管控等方面继续研究。

参考文献:

[1] 王海鹏. 基于 VRML 和 JAVA 的机电一体化设备三维虚拟监控系统[D]. 北京:北京工业大学,2007.
[2] 刘宇轩. 基于 Unity3D 的工业机器人虚拟操作系统的研究与实现[J]. 计算机与数字工程,2017,45(8):1615-1619.
[3] 张涛,唐敦兵,张泽群,等. 面向数字化车间的介入式三维实时监控系统[J]. 中国机械工程,2018,29(8):990-999.
[4] 程立辉,刘大成,金小伟,等. 基于虚拟现实的设备网络化远程监控系统设计[J]. 制造技术与机床,2004(11):86-88.
[5] 张杰峰. 基于 VC 和 Open GL 的工业机器人三维建模[J]. 机械研究与应用,2007,20(2):112-114.
[6] 孔超. 基于虚拟现实的机械安全防护系统研究[D]. 南京:南京理工大学,2017.

收稿日期:2019－07－05