Eurographics Italian Chapter Conference (2007)
Raffaele De Amicis and Giuseppe Conti (Editors)

Human Robot Interaction in Virtual Reality

G. Di Gironimo, A. Marzano, A. Tarallo

University of Naples Federico II, Italy

Abstract

In this paper, we describe a framework which helps designers visualize and verify the results of robotic work cell simulation in a Virtual Environment (VE). The system aims at significantly reducing production costs and error sources during manufacturing processes. The means to achieve these goals are the development of a prototypical VE for the support of robots planning tasks, reuse of animation events, and the implementation of customization tools for animation elements and their behavior. By using advanced Virtual Reality (VR) techniques, the system is also able to direct the focus of the observer to interesting events, objects and time-frames during robotic simulations in order to highlight the Human Robot Interaction within the manufacturing systems.

Categories and Subject Descriptors (according to ACM CCS):
I.3.7 [Computer Graphics]: Animation and Virtual Reality
J.7 [Computers in Others systems]: Industrial Control and Process Control

1. Introduction

In recent years industrial engineering has been oriented towards the development of flexible manufacturing systems, [Cra97] and in particular man-machine interaction systems. Research in robotics is looking for different applications where a human being is to be conceived not exclusively as an operator programming off-line the robot, but rather as a system interacting with the machine by means of different modes, [CC00].

The Virtual Reality (VR) technology offers a highly potential in terms of planning and development of manufacturing systems, [CDM06a], [CDM06b], [DDM06]. In this work, we propose a new research methodology that uses VR techniques in the field of the so-called *Anthropic Robotics*. *Anthropic Robotics* refers to the study of the technologies and methodologies to develop automatic machines that operate services in environments cohabited with the humans, such as cooperating robots, [AAB*06].

The robotic systems, that work in the same environment of humans, have to be endowed with strong characteristics of autonomy, reliability and safety. Indeed, they have to be able to respond to breakdowns, collisions or any unexpected change of the operational scenario and to be able to guarantee the human safety at the same time.

2. The Virtual Reality framework

VR simulations need a specific hardware/software framework, in particular:

- a powerful graphic and calculus system;
- a large screen able to display complex systems in 1:1 scale;
- input devices that allow the user to easily navigate and interact with the virtual scene;
- a software (Simulation Manager) able to manage all the aspects of the virtual simulation, [DMP06];
- a 3D audio output device to increase the realism of the Virtual Environment (VE).

The platform we have chosen as Simulation Manager is Virtual Design 2 (VD2), by vrcom GmbH: VD2 is an extensive tool containing many functions for product development, from the creation of Virtual Environment to assembly simulation, [AB98], or ergonomic analysis. However, one of the most interesting feature provided by VD2 is a Software Development Kit (SDK) that allows the programmer to enhance the basic functionalities of the system by developing external modules that interface with the VD2 kernel. The kernel of Virtual Design 2 consists of three main components, [VA06]: the interaction manager, the device manager and the rendering kernel, see Figure 1.

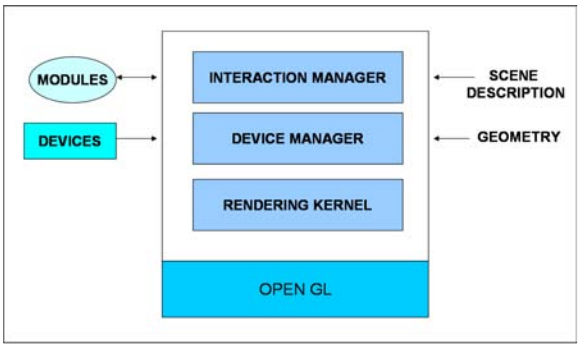

Figure 1: *VD2 kernel architecture.*

The **interaction manager** controls all actions in the Virtual Environment as well as the user' s interactions with the virtual scene. It can be configured with a single *scene description file*, that is a script file which describes the static and dynamic configurations of the virtual objects. The description is based on *events*, *actions* and *objects*. The basic idea is that certain events will trigger certain actions, properties, or behaviors. For example, when the user touches a virtual button, a light will be switched on.

The **device manager** initializes and controls the hardware devices used in the Virtual Environment. It provides the mapping of physical devices to logical devices, as illustrated in Figure 2. This concept simplifies scene building and enhances portability, limiting the concerns about which tracker-system is used or which machine it is attached to. The device manager supports the most common VR devices, like *Spacemouse*, tracking systems (*Polhemus*, *Ascension*, *ART*, *Vicon*, *Intersense*), digital data gloves (*Cyberglove*, *5DT*), and haptic systems (*Phantom*).

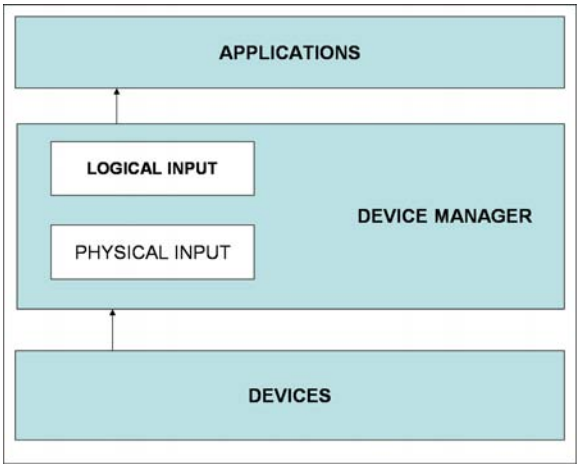

Figure 2: *The VD2 device manager architecture.*

The **rendering kernel**, based on OpenGL, loads the geometry, maintains a hierarchical *scene graph* and renders it. The renderer supports multiple graphics pipes and more than one rendering window per graphic pipe. The rendering kernel also offers several built-in functions for stereo viewing. Stereo viewing can be achieved with dual pipe rendering, shutter, MCO-style, interlaced, or anaglyph (green-red stereo).

2.1. The basic *actions* set

VD2 provides a complete set of commands for planning the behavior of the VE in relation to the user interaction. Many commands describe *actions* that operate on the objects in the VE. Among the supported *actions*, the ones we have used for robotic simulations are:

- **grabbing**: what makes the virtual experience really "interactive" is the possibility to grab virtual objects and to move them through the scene. The "grab" action first makes an object "grabbable". Then, when the hand touches it, it will be attached to the hand.
- **changing object attributes**: these actions allow the user to change some objects attributes, such as materials, visibility, position, etc.
- **sweeping**: this action traces the movement of an object in the VE, by replicating its shape.
- **animations**: some actions allow the user to record and playback the movement of one or more objects of the virtual scene.
- **gravity**: this feature increases the realism of the virtual world, making objects fall in a certain direction and bounce off some "floor objects", that can be specified separately for each object.
- **constraints**: the VD2 kernel allows the user to constrain the movement of an object in the virtual environment. These constraints provide an easy way to define simple interactive kinematics, such as virtual doors and car hoods. By default, when the constrain action is active, the object is linked to the virtual hand, so that it tries to follow the hand's motion but only within the constraint.

2.2. Dynamic Shared Objects

As aforementioned, the features provided by the VD2 kernel can be enhanced by functions defined in external modules, called *Dynamic Shared Objects* (DSO).

Generally, each DSO module contains a set of functions developed for a specific application target, as a *plug-in*. The basic installation of Virtual Design 2 already provides many *plug-ins*, for instance to manage interactive menus or to make snapshots of the virtual scene.

A DSO module is a *dynamically linkable* object file, which allows the linking of the module to the VD2 kernel to be made at run-time, [PW72]. Moreover, the module is

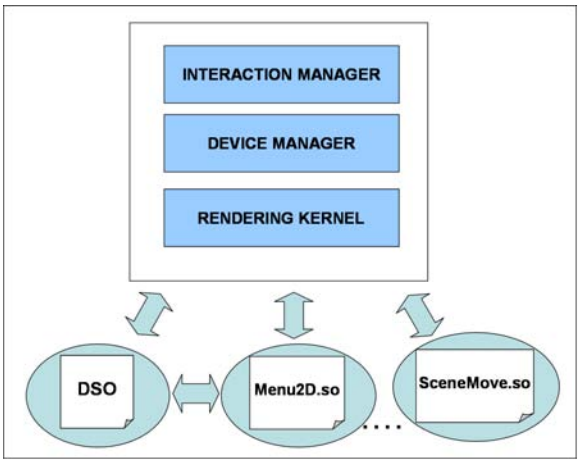

Figure 3: *DSO modular approach.*

shared, meaning that many different processes can share the library functions at the same time, see Figure 3. This modular approach offers three main benefits:

1. The object code is loaded in the physical memory only once and then it can be used by multiple processes via virtual memory management, [Dre06];
2. It is easy to add new features to Virtual Design 2 and maintain them;
3. The object code is linked to the VD2 kernel only when the features implemented in the module are really needed.

The functions provided by DSO modules can be used as well as the basic commands, by specifying them in the *scene description file*.

3. DSO for robot control

The VD2 computational engine does not provide native support for defining and handling kinematic chains. For this reason we developed a DSO module, called *robot.so*, to manage open kinematic chain manipulators in Virtual Reality. The main goals are:

- The plug-in has to be flexible, so that the same functions have to be suitable for different types of robot;
- The robot has to be able to reproduce an user-defined path;
- The user has to be able to manage the robot in real-time;
- Any eventual end-of-stroke condition has to be signaled to the system.

3.1. The robot hierarchical model

In order to use the functions provided by the DSO module, the first step is to arrange the geometric model of the robot. In general, a kinematic chain is a set of rigid elements, called *links*, connected by *joints*. A joint is essentially a constraint on the geometric relationship between two adjacent links. Since VD2 does not provide a really constraint-based Virtual Environment, the *scene-graph* tree structure has been used to keep the logical sequence of the different links. Thus, each joint of the chain is represented by an *assembly node*, as shown in Figure 4.

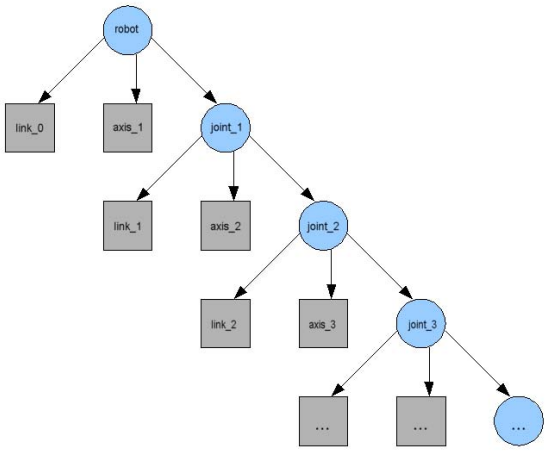

Figure 4: *The robot hierarchical model.*

Thanks to the hierarchical structure of the *scene graph*, the programmer does not have to be concerned about the numerical solution of the direct kinematic problem. Indeed, a single geometric transformation, such as a rotation about an axis, can be defined for a specific joint of the chain, without concerns about the configuration of the other joints. However, since the tree structure of the *scene-graph* is acyclic, the hierarchical model described above is only suitable for open kinematic chains.

3.2. Robot Configuration file

One of the most important goals is the flexibility of the module: in other words the DSO module should be able to handle different types of kinematic chain, independently from number and type of the axes the robot is equipped with. In order to achieve this, the kinematic chain has to be described in a configuration file, which specifies not only names of joints and axes, as defined in the robot *scene-graph*, but also type (revolute or prismatic) and working range of each axis, see Figure 5.

3.3. Robot task planning

Early industrial robots were programmed by moving the robot to a desired goal point and recording its positions in a memory, which the sequencer would read during playback. During teaching phase, the user can guide the robot directly by hand, or through the interaction with a *teach pendant*, that

```
IRB7600.cfg *
# IRB7600 configuration file

# Sintax:
# ROBOT       = int     Robot ID
# JOINT       = string  Name of the joint assembly node, as defined in the scene-graph
# AXIS        = string  Name of the axis geometry, as defined in the scene-graph
# QMAX, QMAX  = float   Axis working range
# TYPE        = int     Joint type (1=REVOLUTE, 2=PRISMATIC)
# Q0          = float   Initial joint position.

ROBOT = 0
{

  JOINT joint_1
  {
      TYPE = 1;
      QMIN = -180;
      QMAX = 180;
      Q0 = 0;
      AXIS = axis_0;
  }
  JOINT joint_2
  {
      TYPE = 1;
      QMIN = -60;
      QMAX = 80;
      Q0 = 0;
      AXIS = axis_1;
  }
  JOINT joint_3
  {
```

Figure 5: *An Industrial robot configuration file.*

is a hand-held control terminal which allows to move each joint of the manipulator, [Cra03].

The DSO module provides functions to simulate in Virtual Reality both aforementioned teaching systems. Indeed, it is possible to control the kinematic chain through the *flystick*, that is a wireless interaction device designed especially for VR applications, or to make the robot follow a tracked object, such as the virtual hand.

Moreover, the set of actions described in the section 2.1, can be specified in the *scene description file*, in order to carry out the robotic simulation as realistically as possible.

3.3.1. Path planning

The DSO module allows to define different postures for the robot, by specifying each of them in the *scene description file*. Moreover, it is possible to handle the kinematic chain in real-time, by relating an input from a VR device, such as a button of the *flystick*, to the handling of a specific joint, see Figure 6.

Figure 6: *Real-time path planning.*

In this way, the user causes the robot to assume the desired posture, by using the *flystick* as a *teach-pendant*. Each posture can be stored in a file, so that the user can define a point-to-point path. The reproduction of the defined path then can be triggered by an event, as it happens for any other action in the VE. Thus, the features described above provide an easy way to plan a collision-free path for the robot task. Furthermore, the integration with the underlying software architecture also allows the user to plan quite complex behaviors, so that the robot can manipulate objects or manage any eventual collision, see Figure 7.

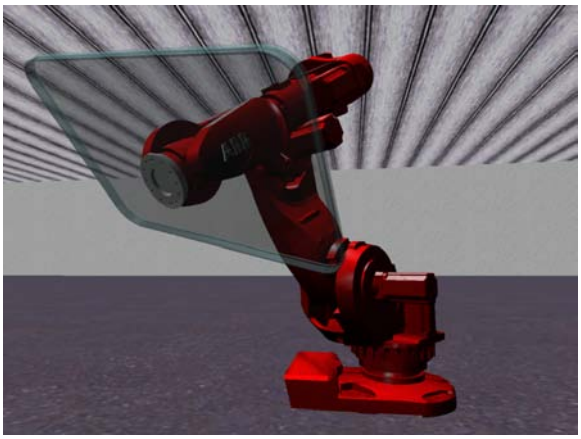

Figure 7: *The manipulator reproducing an assembly task.*

3.3.2. Direct *end-effector* positioning

Generally, finding the joint angles for a given position of the *end-effector* in the operational space requires an inverse kinematic approach, as described in [ZB94]. Since this analysis is limited to open kinematic chain manipulators, a simpler but effective methodology has been adopted.

As aforementioned, VD2 provides a specific *action* that cause a virtual object to follow the user hand, within a specified constraint. For instance, an object can only rotate about a defined axis, according to the movement of the virtual hand. Unfortunately, each constraint is related to a single object in the *scene-graph* and it is treated separately from the other constraints. In other words, the user cannot define directly kinematic relationships among two or more virtual objects.

This approach is suitable for modeling simple kinematics, such as a virtual door, but it can lead to an unexpected behavior when it is applied to a kinematic chain, because generally each link of the chain will move independently from the other elements, as illustrated in Figure 8.

In order to avoid the breaking of the kinematic chain, each constraint operates on a different joint-node of the hierarchical model described in section 3.1, rather than directly on the geometries of each link.

For instance, according to the kinematic chain shown in Figure 9, the first joint-node contains the whole kinematic chain, the second includes only the last two links and finally the third node is just the last link of the chain.

Since all the geometries belonging to a specific joint-node

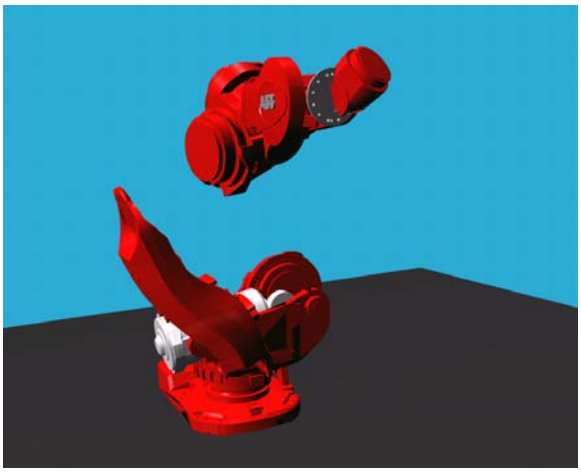

Figure 8: *The kinematic chain breaks during the constrained movement.*

act as a single "rigid object" during the movement, the geometric relationships among the different links will be kept in any case.

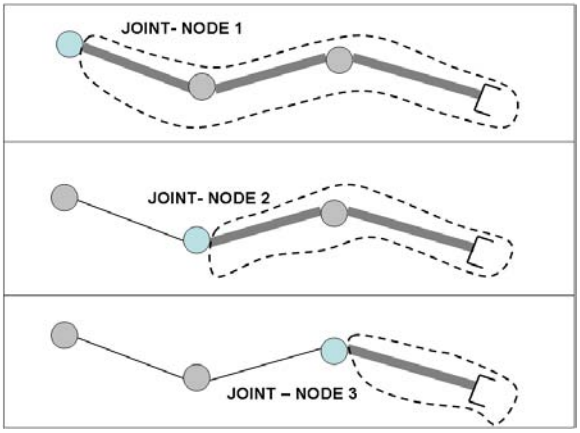

Figure 9: *The hierarchical modeling: the joint-nodes of a kinematic chain.*

Many constraints can be triggered by a single event, such as the collision between the virtual hand and a specific link of the robot. In this way, the user can drag the whole kinematic chain by "grasping" the *end-effector* until the robot reaches the desired position, see Figure 10.

At the same time, it is also possible to move by hand only one or more links of the chain.

4. Conclusions and future work

As aforementioned, the library functions allow the user to easily plan an intended task for any type of open kinematic

Figure 10: *The kinematic chain being dragged by the virtual hand.*

chain manipulator: it is only necessary to prepare the robot *scene-graph* and then edit the configuration file according to the type of chain. Since the DSO module is completely integrated with the underlying software framework, the user can take benefit from all the others functionality provided by the Simulation Manager. For instance, it is possible to display the working area of the robot, highlight eventual collisions between the robot and any object in the VE or trace the path of the end-effector (*sweeping*) during the task execution, see Figure 11.

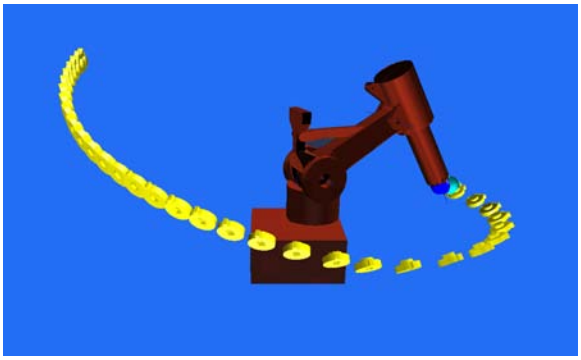

Figure 11: *The sweeping action applied to the end-effector.*

The basic command set can also be used to simulate specific robot behaviors triggered by certain events, eventually generated by external modules. Thus, the modular approach adopted by VD2 kernel could be used to interface a real sensor network with a virtual robot cell. In this way, it would be possible to test the safety strategies adopted to control robots that operate in anthropic domains.

However, the plug-in has some limitations:

- The user cannot define a path in the operational space directly, because the library is not able to perform inverse

kinematic operations. Indeed, the solution of the inverse kinematic problem would break the requirements of flexibility of the DSO module, since it is strictly depending on the particular robot type. Moreover, it would need to consider also eventual kinematic redundancy issues, [SS00];
- The module can manage only open kinematic chain manipulators;
- A dynamic model of the kinematic chain has not been implemented.

Furthermore, in order to use the function set provided by the DSO module, the user has to prepare the robot *scenegraph* and the related configuration file manually. Thus, a future goal will be to develop a graphic *wizard* to lead the user through the configuration process.

5. Acknowledgments

The authors, who have equally contributed to this work, thank the vrcom GmbH for the indispensable technical support and Firema Trasporti SpA for the case study. Further the authors deeply thank Prof. Francesco Caputo and Eng. Stefano Papa for their helpful discussions and suggestions about future works. The present work has been developed with the contribute of POR Campania 2000-2006 - MIS 3.16, performing the activities of the Competence Center for the Qualification of Transportation Systems founded by Campania Region.

References

[AAB*06] ALAMI R., ALBU-SCHAEFFER A., BICCHI A., BISCHOFF R., CHATILA R., DE LUCA A., DE SANTIS A., GIRALT G., GUIOCHET J., HIRZINGER G., INGRAND F., LIPPIELLO V., MATTONE R., POWELL D., SEN S., SICILIANO B., TONIETTI G., VILLANI L.: Safe and dependable physical human-robot interaction in anthropic domains: State of the art and challenges. In *IROS 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems. Workshop on Physical Human-Robot Interaction in Anthropic Domains, Beijing (Chine)* (Oct. 2006). http://www.laas.fr/~felix/publis/pdf/iros06ws.pdf.

[AB98] ABSHIRE K., BARRON M.: Virtual mainteinance: Real world application within virtual environments. In *Proc. of Realiability and Maintainability Symposium, Ohio* (1998).

[CC00] CHEN D., CHENG F.: Integration of product and process development using rapid prototyping and work cell simulation technologies. *Journal of Industrial Technology 16*, 1 (2000), 2–5.

[CDM06a] CAPUTO F., DI GIRONIMO G., MARZANO A.: Approach to simulate manufacturing systems in virtual environment. In *Proc. of the XVIII Congreso Internationa de Ingeniería Gráfica* (May 2006).

[CDM06b] CAPUTO F., DI GIRONIMO G., MARZANO A.: Ergonomic optimization of a manufacturing system work cell in a virtual environment. In *Proc. of 5th International Conference on Advanced Engineering Design* (June 2006). Selected paper for the Acta Polytechnica Journal, forthcoming.

[Cra97] CRAIG J.: Simulation-based robot cell design in adeptrapid. In *Proceeding of the 1997 IEEE International Conference on Robotics and Automation, ICRA, Albuquerue* (Apr. 1997), vol. 4, pp. 3214–3219.

[Cra03] CRAIG J. J.: *Introduction to Robotics: Mechanics and Control*. Prentice Hall, 2003.

[DDM06] DE AMICIS R., DI GIRONIMO G., MARZANO A.: Design of a virtual reality architecture for robotic work cells simulation. In *Proceeding of Virtual Concept 2006, Playa del Carmen, Mexico* (Nov. 2006).

[DMP06] DI GIRONIMO G., MARZANO A., PAPA S.: Design of a virtual reality environment for maintainability tests and manufacturing systems simulations. In *Proceeding of International Conference CIRP-ICME 2006, Ischia, Italy* (July 2006).

[Dre06] DREPPER U.: *How To Write Shared Libraries*. Red Hat, Inc., Aug. 2006. http://people.redhat.com/drepper/dsohowto.pdf.

[PW72] PRESSER L., WHITE J.: Linkers and loaders. *ACM Computers Surveys 4*, 3 (Sept. 1972), 150–151.

[SS00] SCIAVICCO L., SICILIANO B.: *Robotica industriale - Modellistica e controllo di manipolatori*. McGraw-Hill, 2000.

[VA06] VA: *Virtual Design 2 - Programmers Guide 4.5.1*. vrcom GmbH, 2006.

[ZB94] ZHAO J., BADLER N. I.: Inverse kinematics positioning using nonlinear programming for highly articulated figures. *ACM Transactions on Graphics 13*, 4 (1994), 313–336.