SPANviewer - A Visualization Tool for Advanced Robotics Applications

Stefan Kimmer, João Rebelo and André Schiele
Faculty of Mechanical, Maritime and Materials Engineering, Delft University of Technology, Delft, The Netherlands
Telerobotics and Haptics Laboratory, ESTEC, Noordwijk, The Netherlands
Email: {stefan.kimmer, joao.rebelo, andre.schiele}@esa.int

***Abstract*—This paper describes the Space Portable Applications Network viewer (SPANviewer), a visualization tool for advanced robotics applications. The viewer can be used to visualize structured kinematics and combinations thereof with video streaming. Some examples of the viewer applications such as control algorithms debugging, teleoperation with augmented reality and/or 3D stereoscopic view and analysis of low bandwidth video transmission in teleoperation are highlighted in this paper.**

***Index Terms*—Visualization, 3D, Stereoscopic, METERON, Simulation, Viewer, Robotics, Network, SPAN.**

I. Introduction

When developing control algorithms and systems for teleoperation, the need often arises for real-time debugging of new kinematic and dynamics algorithms, image recognition techniques and real/virtual image combination for teleoperation scenarios. Currently, no tool exists that allows a quick and adaptable definition of robotics structures that can be controlled in real-time and potentially serve as an overlay of real video images. The existing visualization tools for virtual reality focus on realistic image rendering but lack the specific robotics-oriented applications (e.g. structure kinematics representation), whereas robot simulators (e.g. Webots [1]) aim at emulating real-hardware and dynamics behavior which makes them very complex and reduces real-time capabilities.

The Space Portable Applications Network (SPAN) aims at being a set of highly reusable tools and libraries which can be used to allow quick implementation and debugging not only of new algorithms but also of more complete system implementations. The goal of this paper is to describe the visualization tool of this library, the SPANviewer, detailing its software structure and design while highlighting the use-cases and applications for which it was developed.

The SPANviewer allows the user to define any robot by a hierarchical XML (Extensible Markup Language) structure using parameters of standard representations common in robotics (e.g. Denavit-Hartenberg or frame coordinate system). The structures can be realistically represented using CAD (Computer Aided Design) models, but simple built-in models can be used instead if CAD models are not available. The XML input file also allows the definition of camera view-points in mono and stereoscopic view. Both the structure joint positions and the camera view-points can be adjusted in real-time through a UDP (User Datagram Protocol) or Shared Memory communications interface. In addition, the viewer is able to render a real-time video stream and overlay it with the virtual structures controlled in real-time. Given the need of using both Windows and real-time Linux-based operating systems, the viewer implementation is platform independent.

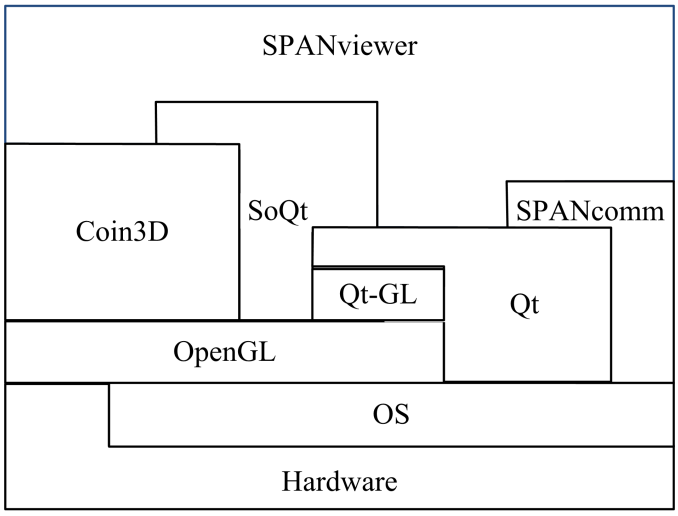


Fig. 1. Overview of the stack of used libraries. The OS can be any of the operating systems supported by Qt and OpenGL, e.g.: Microsoft Windows XP, 7, 8 or Linux

This paper is organized as follows: Section II describes the software architecture and design, Section III highlight the main applications of the viewer and the conclusions and future developments are stated in Section IV.

II. Software overview

As explained in the introduction, the SPANviewer aims at being platform independent. For this reason it was implemented in C++ on top of Coin3D [2], a higher level library of the OpenGL [3] library. For windowing and other graphical user interface functionality the SoQt binding to the Qt library [4] is used. A diagram of the component stack is shown in Fig. 1.

Since all of the components are cross platform the SPANviewer is running on all major platforms. For newer Windows and Debian-based Linux distributions an installer and a package exist that enables fast deployment of the software. This section describes the main characteristics, such as the scene

```
<?xml version="1.0"?>
<world xmlns="ESA-Telerobotics-Lab-SPAN">
  <scene units="0.2">
    <structure convention="dh">
      <joint a="0" alpha="-1.5708" d="0.5" offset="0
          " theta="0">
        <joint a="0.5" alpha="1.5708" d="0" offset="
            0" theta="0">
        </joint>
      </joint>
    </structure>
  </scene>
</world>
```

Listing 1. SPANviewer example model file to generate a series structure. The structure is defined under the <scene> node using the DH convention. The visualization is visible in Fig. 2(a)

```
<?xml version="1.0"?>
<world xmlns="ESA-Telerobotics-Lab-SPAN">
  <scene units="0.2">
    <structure convention="dh">
      <joint a="0" alpha="-1.5708" d="0.5" offset="0
          " theta="0">
      </joint>
      <joint a="0.5" alpha="1.5708" d="0" offset="0"
          theta="0">
      </joint>
    </structure>
  </scene>
</world>
```

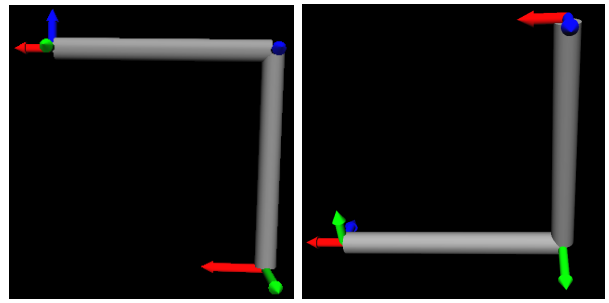
Listing 2. SPANviewer example model file to generate Fig. 2(b). It is an XML- file which follows the XML schema under the namespace ESA-Telerobotics-Lab-SPAN. The parallel structure is defined under the <scene> node using the Denavit-Hartenberg convention.

model, the camera model and the available communications mechanisms. Finally some of the additional capabilities of the SPANviewer, such as the stereoscopic view and the 3D model rendering are also presented.

A. Scene definition

At the base of the viewer functionality stands the world model described in XML where the scene and cameras can be defined. This XML is described by the ESA-Telerobotics-Lab-SPAN schema. Under the <scene> tag all the objects and structures are defined. The viewer allows the simulation of both series and parallel structures defined by Denavit-Hartenberg parameters (using the convention in [5]) . An example of a parallel and a series joint structures model is shown in List. 1 and the output in Fig. 2. The scenes resulting from these models are depicted in Fig. 2. Note that the only difference between implementing a series and a parallel structure is on the hierarchy of the XML model.

Besides the robotic structures it is also possible to render pre-defined objects such as cubes, spheres, cylinders and arrows and to include custom position and rotation offsets. As already mentioned, the viewer is independent from any physics or kinematics engine, therefore it is the task of the user to implement the needed algorithms and send the actual joint values to be displayed through the viewer communications channel.



(a) Series joint (b) Parallel joint

Fig. 2. Example of simple series and parallel structures. Note that in both images the center point is represented by the axis on the lower right corner. In this scene, the default built-in models are used to represent the structure. The model files which can be used to generate this can be seen for (a) in List. 1 and for (b) in List. 2

```
<?xml version="1.0"?>
<world xmlns="ESA-Telerobotics-Lab-SPAN">
  <camera>
    <monocamera window_title="monocamera X">
      <pose x="100" y="0" />
    </monocamera>
    <monocamera window_title="monocamera Y">
      <pose x="0" y="100" />
    </monocamera>
    <stereocamera>
      <pose x="700" z="200" z="0" />
      <screendef diameter="1016" />
    </stereocamera>
  </camera>
  <scene units="100">
    <cube sx="100" sy="10" sz="20" tx="200"
        axis_scale="5"/>
  </scene>
</world>
```

Listing 3. Example with three cameras viewing a cube. The three cameras will produce four windows, one for each mono-camera and two for the stereo-camera. The two windows of the stereo camera can be placed to two projectors of a passive stereoscopic viewing system. The resulting windows are shown in Fig. 3 and in Fig. 4

The viewer is able to import model parts written in the Virtual Reality Modelling Language (VRML). Most computer aided design program have the possibility to export their formats to VRML into a world file (*.WRL file). The WRL file can then be included at any point in the scene or at any joint of a structure.

Images which can be used to display a video stream can be added to a scene. This can be a shmimage object as in List. 4.

B. Camera definition

With the SPANviewer it is possible to visualize a scene with predefined point of views. Each point of view is thereby displayed in a separate window. The location of the appearance of each window can be configured for multi monitor systems. Hence it is possible to set up multiple view point human machine interfaces. In List. 3 two viewpoints are configured with the <monocamera> tag. One camera enables the view of the back of a cube and the other a side-view of that cube. The resulting windows are shown in Fig. 3.

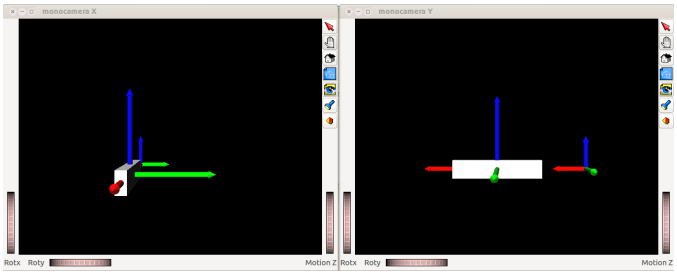


Fig. 3. Two windows of two mono-cameras as defined in List. 3. The left window shows the cube from behind while the right window shows the same cube from the side.

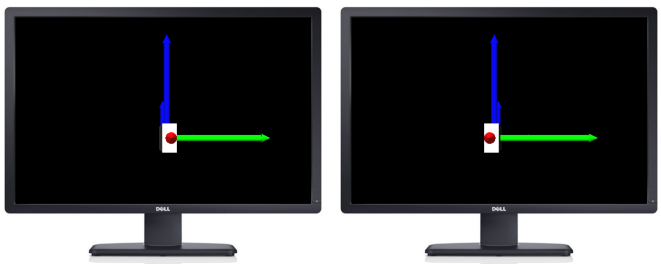


Fig. 4. Two windows of the stereo-cam output of List. 3. The left window represents the view of the left eye and the right window for the right eye respectively.

It is also possible to view a scene in a stereoscopic mode. Therefore two viewpoints will be created which represent the view of the left and right eye of the human. Each side can be displayed as a separate window which allows the usage of passive stereoscopic displays. A passive system is often used with larger projections where each of the two projectors needs its own image source. A stereo-camera has been defined besides two other mono-cameras in List. 3. This stereo-camera results in the two windows as shown in Fig. 4. Those two windows can be viewed by a passive stereoscopic display system. There a human observer will perceive the cube 200mm in front of the screen. The latter will be true if the observer is located 700mm in front and 200mm above the center of the screen because the pose of the stereocamera is configured as x="700" and z="200" (the global unit is assumed to be mm). The screen size can be configured as the diameter in a screendef tag. In this example it is set to a screen with a 40" (1016mm) diagonal. The cube will appear in the size it is specified in the XML model file, i.e 100mm long, 10mm wide and 20mm height.

C. Communication Model

Another key functionality of the viewer is the ability to enable fast configurable communications with the model. To achieve this goal the user can, in the XML model file, reference to different communication channels that will receive new values for the objects characteristics. These characteristics can be e.g. position and orientation.

A general way of using different channels is shown in List. 4. There the images could be used to display a video

```
<?xml version="1.0"?>
<world xmlns="ESA-Telerobotics-Lab-SPAN">
  <camera>
    <monocamera channel="1" />
    <stereocamera channel="k" />
  <camera>
  <scene>
    <structure channel="1" />
    <structure channel="m" />
    <shmimage channel="1" />
    <shmimage channel="n" />
  </scene>
</world>
```

Listing 4. Example model file of the SPANviewer with the child nodes configured to listen on different communication channels. The letters k,m and n must be hereby be replaced by a valid identification number. A schematic plot of this configuration is shown in Fig. 5. The exact communication mechanism (the definition of the channels) is configured only at execution time. An example is shown in List. 5.

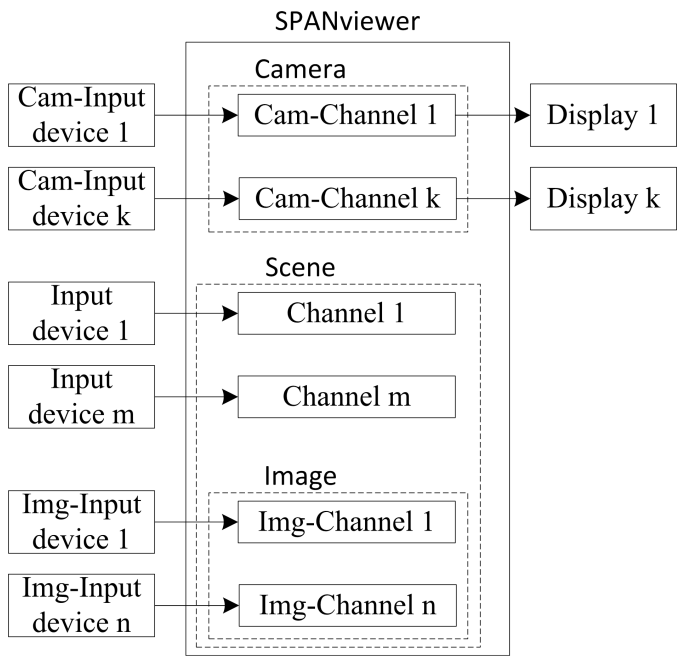


Fig. 5. The general communication model of the SPANviewer. A communication category for general, camera image input devices exists. Each camera will result in a new display (window).

stream written in shared memory, the view-points of the virtual cameras could be controlled by a tracking mechanism and the structure could be controlled by an exoskeleton. The communication channels are grouped to enable different update speeds for each group. Those groups are: camera-channels to allow view-point updates, channels to allow scene object updates and image-channels to allow video streaming. An overview of the communication groups and how they are embedded in a SPANviewer model is given in Fig. 5. Inside a model file only the identification number of the channel can be configured, the channels itself can be configured at the command line options of the viewer. Each channel can be a UDP port to listen to or a shared memory key to read from. In the command line options a pure numerical argument represents a UDP port,

```
spanviewer --input xml.xml --channel 25000 shm --cam
    -channel shmCam 18000 --img-channel shmImg1
    shmImg2
```

Listing 5. Example command line to load the xml.xml model file. For each communication category two channels are defined. Whereas the first channel get the identification number 1 and the second the identification number 2. The channel can be addressed by those identification numbers in the xml file.

```
<structure convention="dh" channel="1">
  <joint a="10" alpha="0">
    <joint a="20" alpha="1.5708" />
  </joint>
</structure>
<cube channel="2"/>
```

Listing 6. Example of objects of a SPANviewer model file. The structure contains two joints which can be controlled through communication channel 1. In addition a cube is located in the scene and can be controlled through communication channel 2.

others represent a key for a shared memory region. In List. 5 an example is shown where various channels are defined. During execution each category is checked for changes at a periodic scheduled basis. For instance the image input data can be read with 25Hz while the camera input data can be read with 100Hz. This allows for a configurable and transparent way of prioritization to the different input device categories. If the viewer is unable to render in the desired speed a message will be displayed about the dropped input data frames. In the case input data is dropped the viewer renders as fast as possible. The communications are done by simple arrays of doubles and the order is identical to the one in which the objects are defined in the XML. This order can be printed to the command window when the SPANviewer is started for user reference. In List. 6 a more concrete example is shown which contains a manipulator with two joints and a cube in the same scene. These two objects can be controlled through two different channels. The joints of the manipulator can be controlled by, e.g. a joystick through channel 1 and the cube can be controlled, e.g. by a computer simulation through channel 2. The model can be executed with:

```
spanviewer --input in.xml --channel 25000 shm
```

Then the joystick must send its data to UDP port 25000 (channel 1) and the computer simulation must write the data to the shared memory region with the key "shm" (channel 2). For test purposes values can be send with the enclosed sendvalues program. If for example the angle of joint 1 shall be 1.1 and the angle of joint 2 shall be 1.2 following python script can be used:

```
import socket, struct
sock = socket.socket(socket.AF_INET, socket.
    SOCK_DGRAM)
sock.sendto(struct.pack("dd", 1.1, 1.2), ("127.0.0.1
    ", 25000))
```

III. Applications

The SPANviewer is designed to be usable in various fields of robotics, nonetheless the main driver of these developments is the Multi-Purpose End-To-End Robotics Network (METERON) project [6]. In the next subsections three of the applications which are targeted specifically at METERON or related developments are outlined.

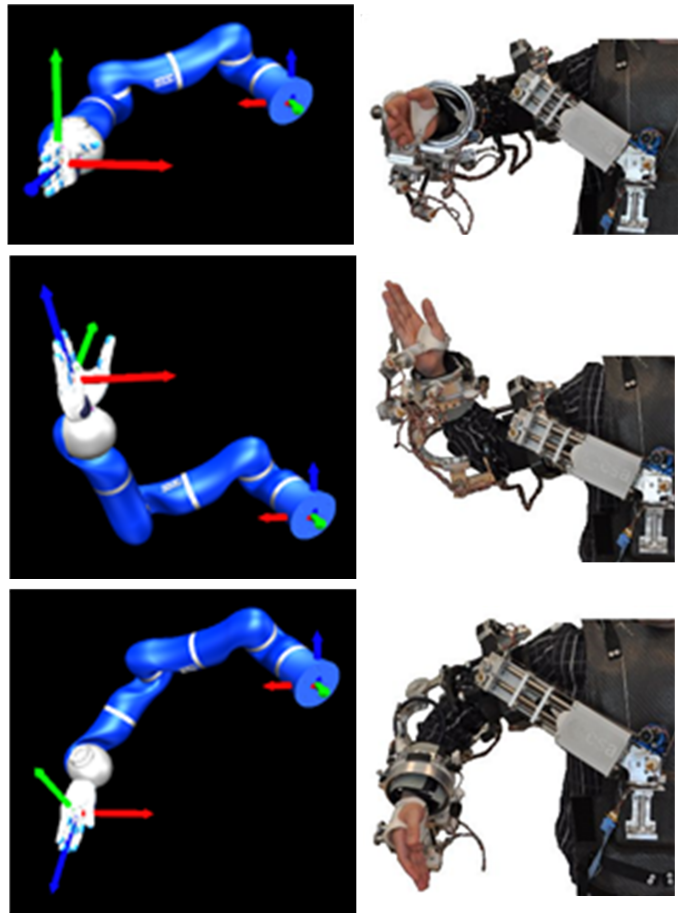


Fig. 6. Example of kinematic tracking debugging.

A. Kinematic Studies

One of the main applications of the SPANviewer is to allow quick debugging and analysis of different types of control algorithms. For example, when controlling a 7 degree-of-freedom manipulator using an exoskeleton, the visualization is needed to ensure that the robot behavior is following exactly the commands given by the exoskeleton. Using the SPANviewer, a kinematic structure based on the same D-H parameters used for the kinematics computation can be defined. The kinematic algorithm (such as e.g. Jacobian transpose augmented with elbow control [7]) can be easily analyzed by sending the resulting joint values in real-time to the visualization tool. An example of simultaneous views of the exoskeleton and the simulation is shown in Fig. 6

It is also common in telerobotic applications that the user is provided with views from fixed cameras in different angles. To simulate these scenarios, the viewer allows the simultaneous implementation of different viewpoints. An example of a typical scenario with Top, Left and Front views of the LWR robot is shown in Fig. 7. This experimental setup was

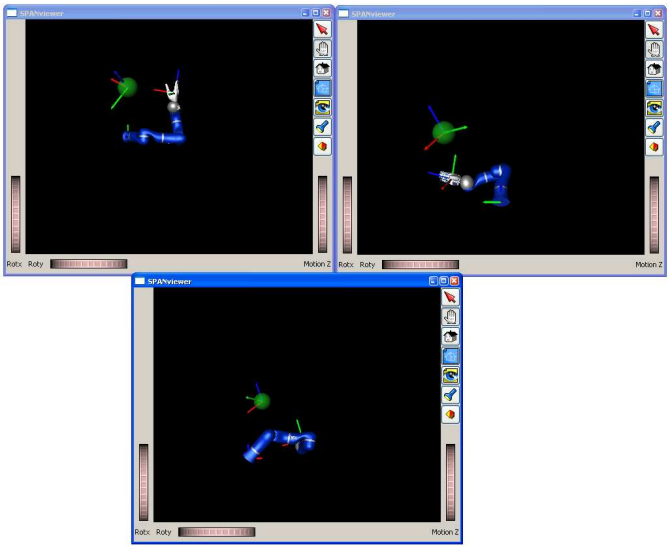


Fig. 7. SPANviewer showing three simultaneous views of the LWR

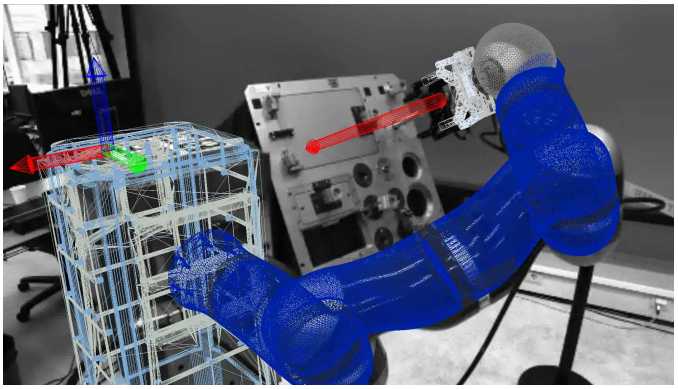


Fig. 8. Overlay of a real scene with a virtual scene. The real scene is a gray-scale video stream from a camera. The overlay is a model of the robot used in the real scene while seven joints are actuated. Additionally the force which is applied to the robot is displayed at the tool of the robot.

used to experiment on the effects of perception on the user performance on a positioning task.

B. Overlay of Video Streams

In teleoperation scenarios, in particular in space teleoperation, it is common that the available bandwidths are very limited which allows only low-quality video to be transmitted. Naturally, this leads to extended difficulties in executing remote tasks. Nonetheless, since models of the manipulator and potentially of the environment can be available it is possible to enhance the video perception to the user by overlaying virtual scenes on top of the real video stream.

The SPANviewer allows simultaneous display and real-time update of both real and virtual scenes. The robot structure is identical to the one described in Fig. 8.This overlaid virtual robot gives the operator additional information about the structure position which allows a better evaluation of the current situation at the remote site. It is also possible to render

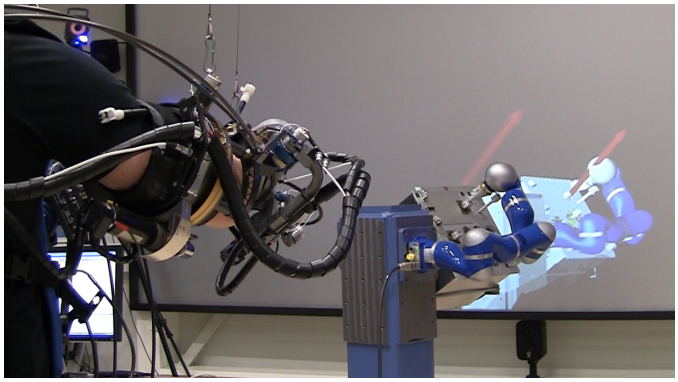


Fig. 9. Example of an augmented reality in front of a passive 3D-screen. The robotic arm is controlled by the operator which is wearing an exoskeleton. The head of the operator is tracked which enables the display of an arrow at the point in the three dimensional space where the corresponding force is exerted. In addition a copy of the robot is visualized behind and to the right of the real one.

additional items, e.g. color-coded arrows representing forces and velocities, which can transmit to the operator a complete and intuitive overview of the operations scenario within a single display.

The virtual scene can be send to the viewer in predefined formats. The cross platform GStreamer [8] library can be used to stream various video sources to the desired image object inside the viewer. This achieved by the sending the video data to a identified shared memory. Any image object with that identifier of the running model of the SPANviewer will then be displayed with a configured frame-rate.

C. Stereoscopic Visualization and Augmented Reality

It is often desirable to view an object from different viewpoints. If for instance an object has to be touched which lies in a three dimensional space, only one view point is not enough to determine its position in that space. For an intuitive solution the head of the user can be tracked and the point of view of the model will be adjusted accordingly. This results in static model in front of the displaying screen. The tracker values can be sent to the viewer in the same way as other values. A correct representation of such a model is dependent of the size of the displaying device. This can be configured within a SPANviewer model file.

The ability of visualizing objects in a real three dimensional space allows for a mixture of real objects (in front of the displaying device) with virtual objects. In Fig. 9 a copy of a real robot is displayed as a virtual object besides the real robot with determined and configurable geometric relationship in real space.

In addition the applied force to the robot is displayed as a red arrow which pops-up at the point of exertion in real space. If the operator is moving the appearance of the arrow will remain at the same point in space.

IV. Conclusion and Future Work

As described in this paper, the SPANviewer is currently in a state in which it can already be used to execute several tasks,

ranging from control algorithm debugging to telerobotics operations with virtual overlaid information. The tool has been widely used in the activities of the ESTEC Telerobotics and Haptics Lab. Nonetheless, to further increase its usability and task range it is now planned to integrate the viewer with physics simulation software in which the same description language can still be used to create both the virtual scene and the virtual dynamics world. Also, the set of formats available for the video stream is currently very limited. To enable a more generic integration to other formats, the GStreamer library may be integrated into the SPANviewer. The entire range of SPAN tools aims at being self-contained and run on a distributed system. Currently the simple communication mechanisms available are adequate for direct interaction between two or three different systems. However, as more systems need to interact to create the simulation and visualization environments the simple communications techniques become unmanageable. The Data Distributed Service (DDS) communication middleware will be integrated into the viewer and future SPAN tools to allow a flexible and easy communication between each process. Once these tools and their documentation reach a high maturity level, they can be made available in open-source form to the community for further usage and enhancement.

REFERENCES

[1] Cyberbotics, "Webots," http://www.cyberbotics.com/overview, Apr. 2013.

[2] Kongsberg Oil & Gas Technologies, "Coin3d," http://www.coin3d.org, Apr. 2013.

[3] The Khronos Group, "Opengl," http://www.opengl.org/, Apr. 2013.

[4] Digia Plc, "Qt," http://qt.digia.com, Apr. 2013.

[5] B. Siciliano and L. Sciavicco, *Modelling and control of robot manipulators*. Springer-Verlag, 2003.

[6] A. Schiele, "METERON - validating orbit-to-ground telerobotics operations technologies," in *11th Symposium on Advanced Space Technologies in Robotics and Automation, ASTRA 2011*, 2011.

[7] J. Rebelo and A. Schiele, "Master-slave mapping and slave base placement optimization for intuitive and kinematically robust direct teleoperation," in *International Conference on Control, Automation and Systems (ICCAS)*, 2012.

[8] GStreamer, "Gstreamer," http://www.gstreamer.com/, Apr. 2013.