Modelling and Interfacing Remote Virtual Robots

Zoe Doulgeri, Nikos Zikos and Anastasios Delopoulos
Department of Electrical and Computer Engineering
Aristotle University of Thessaloniki - Greece
Email: doulgeri@eng.auth.gr, nzikos@auth.gr, adelo@eng.auth.gr

***Abstract*— A formal generic model for simulating the kinematic behavior of virtual robotic arms is presented in this work. We introduce an algorithmic procedure, called Virtual Robot Simulation (VRS) Engine, for updating the trajectory of the virtual robot when motion commands are presented for execution. The trajectory itself is modelled as a list consisting of time intervals where robot joints follow a piecewise polynomial path. In addition, the same engine is equipped with the functionality of responding to position requests at arbitrary time instances. Constraints imposed by the operation of VRS in real-time mode are also explored. User interfaces to the virtual robot - including remote access through data networks - are also modelled and the effects of possible communication delays are explored. The combination of the proposed simulation environment with a synchronized real robot is proposed as a means to overcome the lack of direct visual feedback in tele-robotic applications.**[1]

Index Terms - Telerobotic, Virtual Reality, Robot Simulation, Real Time Environment

I. INTRODUCTION

Robotic technology is young but it is growing at a fast pace. Robotic systems have already been extensively applied in factory automation, space exploration, surgery, and military services and soon there will be robots in every house and business. Naturally, as in the computer case, the remote operation and control of robots is going to be a highly desirable feature. So, eventually and similar to the Internet, a robonet will be developed where robots are connected to computers, other robots and humans.

For the time being the technology allows the remote control of robots referred to as teleoperation. Teleoperation can take several forms and can be done via any communication medium while recently researchers, motivated by the Internet's availability, widespread access and low cost, have focused on the Internet based teleoperation (see e.g., [1], [2], [3], [4] and [5]). Stability, transparency and synchronization are among the main features that are desired in any teleoperation system despite the time delay introduced by the communication channel. Ensuring these features specifically in Internet based teleoperated robots is a challenging task. For example, Oboe and Fiorini in [5] proposed the design of a robotic controller that copes with the time delays of the network by modelling its behavior as time varying random process.

In robot teleoperation there is no predefined path; the path is generated in real time by the operator based on the feedback received. The feedback necessary to control a robot is primarily visual but the transmission of images from the operational site places heavy demands on the communication system. As an alternative a local virtual robotic system (VRS) representing the real one can be frequently updated with a relatively small amount of information (e.g. joint angles) that can be transmitted rapidly to enable speedy update of the VRS, [6]. Likewise the effects of control commands can be rapidly demonstrated in the VRS than would be possible by the transmission of images from the distant site. Kuk-Hyun Han et al in [3] argue that time delays introduced by the Internet is difficult to measure and/or predict and they adopt the above idea of a local VRS that is mimicking the motion of its real counterpart. A so called posture estimation scheme is applied in order to keep discrepancies between the position of the real and the virtual robots as low as possible; however in their setup posture of the controlled robot refers to the $[x,\, y]$ location and the direction (θ) of the (mobile) robot. Belousov et al in [4] are also considering the coupling of a local VRS to the remote real robot which in this case is a PUMA robotic arm. Three dimensional VR interfaces are used to (re)present the position of the virtual arm in real time.

In this work we discuss the modelling of and interfacing to remote virtual robots and propose the construction of a simulation engine that is event driven rather than time driven and thus is able to reduce or even eliminate time delay effects. The emphasis of our work is on two major aspects of the VRS: (a) The formal definition of its internal state update procedure that is driven by the issued motion commands. (b) The modular construction of interfaces to the VRS. In fact, two very simple interfacing modules - the Designer and the Observer - are formally defined; a variety of advanced interfaces including those operating over the Internet and/or in play-back mode are constructed using them as building blocks. Both (a) and (b) are exploited next in teleoperation scenarios.

II. SIMULATION ENGINE

The virtual robot simulation (VRS) is an internet based application for the kinematic simulation of a specific robot. The simulation engine (SE) is able to import user motion commands and export the needed position data which through a visualization tool can inform the user for the robot position at every required time instant. The VRS is designed to accommodate a number of simulation scenarios that depend on its use that can range from task planning to the virtual representation of a real robot operation.

The heart of the simulation engine is composed of two main functions. The *Trajectory Generation Function* and the *Current Position Function*.

[1]This work was partially supported by the Greek Ministry of Education within ΕΠΕΑΕΚ/ΠΥΘΑΓΟΡΑΣ − 2 framework.

The *Trajectory Generation Function* is a recursive function that is triggered each time a new motion command is issued by the VRS robot operator:

$$\tau_{k+1} = f(\theta; p_{k+1}, t_{k+1}, v_{k+1}; \tau_k) \quad (1)$$

θ : is a vector of kinematic and kinetic robot parameters that are invariant under the simulation procedure. Such parameters include for example link lengths, maximum joint velocities and accelerations etc.

$(p_{k+1}, t_{k+1}, v_{k+1})$: are the position, time stamp and velocity parameters of a motion command. In particular, $p \in \Re^6$ is the desired position and orientation of the robot's end-effector. The time stamp parameters $t = \begin{bmatrix} t^{iss} \\ t^{exec} \end{bmatrix}$ include the command issue time t^{iss} and the command execution time t^{exec}. The need for such a distinction will hopefully become clear later on. Last, the velocity parameter $v_{k+1} \in [0, 1]$ is a scalar expressing the percentage of the nominal linear and angular end effector velocity that is desired for the motion segment.

τ is the trajectory generated under all the motion commands that have been issued so far (until step k). The trajectory can be viewed as a dynamic list of vectors

$$\tau_k = \{s_1 s_2 s_3 ... s_l\} \quad (2)$$

each describing a trajectory segment. Each motion command may modify old trajectory segments and add one or more new trajectory segments of the form:

$$s_i = \begin{bmatrix} t_{si} \\ t_{fi} \\ a_i \end{bmatrix} \quad (3)$$

where t_{si} , t_{fi} is the start and end time of the time interval the position trajectory segment is valid and a_i is a vector of the coefficients of the polynomial $p_i(t - t_{si})$ characterizing this trajectory segment. The polynomial's constant part, represented by coefficient a_i^0, is the robot's position at the start time of the trajectory segment. For a continuous trajectory $a_{i+1}^0 = p_i(t_{fi} - t_{si})$. For k=0, s_0 contains the initial robot position that can be set to a default position for e.g. zero joint variables.

The *Current Position Function* is a function returning the robot position at a specific time instant t and is a function of the form:

$$q = g(\theta; \tau_k, t) \quad (4)$$

This function finds the trajectory segment that is valid at the required time instant, and then calculates the robot's position using this segment's polynomial coefficients. The position returned to the user is given as a nx1 column matrix which contains the joint angles of the joints or as a 3x1 vector of the XYZ tool position and a 3x1 vector of the tool's Euler angles.

A. Structure of the Simulation Engine

The trajectory designer uses internal speed and acceleration parameters to implement the trajectory generation function given a new motion command. The result modifies the content of the *Trajectory Timeline* $\tau_k = \{s_1 s_2 s_3 ... s_l\}$ that contains all the trajectory segments in the simulation time horizon. The kinematics block uses robot kinematic parameters to solve the robot inverse or forward kinematic problem given the tool position or the joint angles at a time instant. The input data are taken from the trajectory timeline. The block returns to the user the robot joint angles or the Cartesian position and orientation of its links. The Command analyzer block accepts user commands and is responsible for checking their integrity and syntax. It is further responsible for recognizing the command type and route it appropriately.

Commands can be motion commands, current position commands or commands that change the internal simulation parameters. Commands that change the internal simulation parameters specify the parameter and its new value and return nothing to the user. Current position commands return the robot position and the time stamp at which the robot position was requested. This time stamp is an instance in the robot simulation timeline. Motion commands specify the desired end effector position and orientation and/or desired motion type (motion in straight line or joint interpolated motion, [7]) and do not return anything to the user.

An important characteristic of a motion command in the proposed VRS is its time stamp. In actual robotic systems an issued motion command is executed instantly. This may not however be true for robots controlled over the internet. Variable delays may be present in this case and consequently issuing and execution time of motion command may differ considerably. In the proposed VRS the possibility to define different issuing and execution times is given through the motion command time stamp $t = \begin{bmatrix} t^{iss} \\ t^{exec} \end{bmatrix}$. Issuing time t^{iss} is fixed automatically when the user registers the command. Execution time is the time the motion should be executed. It is either defined by the user or it is decided on the basis of a specific policy described in Section II-C.

B. Trajectory Timeline Modification

The trajectory timeline contains the whole robot motion in the time horizon under study. The way a new motion command affects the contents of the trajectory timeline depends on the relation of the execution time with respect to the end times of the trajectory segments.

Let the motion command be: $(p_{k+1}, t_{k+1}, v_{k+1})$. The motion command execution time t^{exec} extracted from t_{k+1} is compared to the end times of the scheduled segments. Then the trajectory is modified with respect to all these trajectory segments s_i with end time greater than the execution time $t^{exec} < t_{fi}$. If the execution time is greater than the end time of the last segment $t_{fl} \leq t^{exec}$ then the execution time is used as the start time of the first new trajectory segment and new trajectory segments are in this case appended on the trajectory list. In the general case, say n scheduled trajectory segments have end times greater than the command's execution time i.e.:

$$t^{exec} < t_{fi} \qquad i = m, ..., m + n \quad (5)$$

where m denotes the order of the first segment affected by the new command. After the insertion of the new trajectory

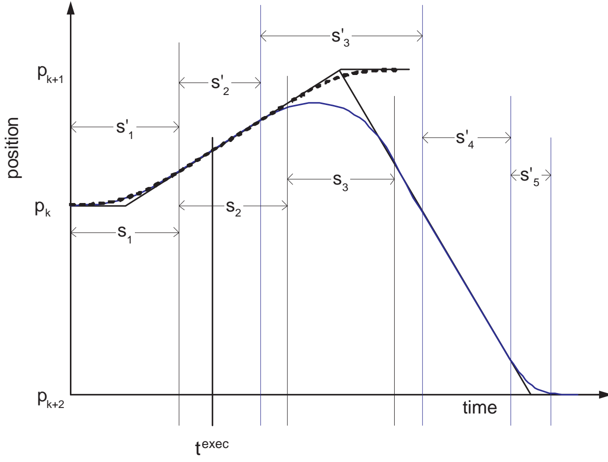

Fig. 1. Trajectory modification.

segments, that replace and expand some old ones the rest of the segments are just time shifted without any change in their corresponding polynomial coefficients. This means that a new position command modifies the trajectory only locally.

Let us for example assume that the trajectory used between two position values (p_k, p_{k+1}) is a linear trajectory with parabolic blends, [8], [9] like the one illustrated by the dashed line in Figure 1. Such a trajectory corresponds to three trajectory segments:

$$\begin{aligned} \tau_k &= \{\dots s_1\, s_2\, s_3\} \\ &= \left\{ \begin{array}{cccc} & t_{s1} & t_{s2} & t_{s3} \\ \dots & t_{f1} & t_{f2} & t_{f3} \\ & a_1 & a_2 & a_3 \end{array} \right\} \end{aligned} \quad (6)$$

with $t_{s2} \equiv t_{f1}$ and $t_{s3} \equiv t_{f2}$.

Let us also assume that the execution time t^{exec} for moving to the new position p_{k+2} falls within the time interval of the linear trajectory segment, i.e. $t_{s2} < t^{exec} \leq t_{f2}$. Then, the new trajectory time line is

$$\begin{aligned} \tau_{k+1} &= \{\dots s_1\, s'_2\, s'_3\, s'_4\, s'_5\} \\ &= \left\{ \dots s_1 \left\{ \begin{array}{cccc} t'_{s2} & t'_{s3} & t'_{s4} & t'_{s5} \\ t'_{f2} & t'_{f3} & t'_{f4} & t'_{f5} \\ a'_2 & a'_3 & a'_4 & a'_5 \end{array} \right\} \right\} \end{aligned}$$

with $t'_{s2} \equiv t_{s2}$.

Trajectory segment s'_3 smoothly changes the end effector's velocity in order to connect the two straight line trajectories s_2 and s'_4 [8]. The new trajectory corresponds to the solid line in Figure 1.

C. Real-time Mode

Up to this point the trajectory, τ_k, is considered as aligned to a time-line with an arbitrary start $t = 0$. In our view this is a powerful abstraction that adds flexibility whenever the proposed simulation engine is employed in experimentation, planning or teaching environments. On the other hand, when the same simulator is to be considered as an emulator of a real robotic system, e.g., when it is used as a synchronized copy of an actual robot, its time-line should be aligned to some real-time clock. As a consequence the following two *causality restrictions* apply:

1) The time-stamps of all motion commands should satisfy the inequality $t^{exec} \geq t^{iss}$
2) Motion commands $(p_{k+1}, t_{k+1}, v_{k+1})$ that, according to the analysis of Section II-B, would result in modifications of the trajectory at time points preceding the real clock time t_c, should be aborted or postponed (depending on the adopted policy). Postponing a command is equivalent to altering its t^{exec} to a value greater or equal to the t_f of the current interval. (Note that the procedure of Section II-B will apply for the computation of the new trajectory for the "postponed" t^{exec}).

However, the causality assumption does not impose any restriction to the time parameter t of Current Position Function $q = g(\theta; \tau_k, t)$, other that it is now referring to time values corresponding to the real-timeline.

III. Interfaces

Users may communicate to the VRS by means of two elementary interfacing modules, namely the *Designer* and the *Observer*. These two modules essentially correspond to, and are able to invoke, the *trajectory generation* and the *current position* functions respectively.

The designer module is responsible to modify virtual robot's trajectory by issuing motion commands of the form $(p_{k+1}, t_{k+1}, v_{k+1})$. In practice, these commands are expressed in a high level language (in our experiments we adopted a VAL-II like syntax) that have though an one-to-one mapping to $(p_{k+1}, t_{k+1}, v_{k+1})$ triplets. Although, in general, the execution time-stamps within t_{k+1} may refer to any time instance, certain causality restrictions apply when the module operates in real-time mode (ref Section II-C).

The observer module is responsible to acquire virtual robot's position by issuing commands corresponding to invocation of *current position function* at any arbitrary time instance t. In practice, a high level syntax is employed for expressing these commands.

The links of the simulation engine to the two elementary interfaces is shown in Figure 2.

The previous elementary modules are the building blocks for user interfaces of higher and differentiated functionality. Of particular interest are the *Supervisor* and the *Viewer*.

The supervisor is the combination of the designer and observer modules. It uses designer's capabilities for specifying robot's trajectory and utilizes observer for collecting feedback from the virtual robot.

On the other hand the viewer is a special instantiation of an observer that is scheduled to repetitively acquire position information at user defined time intervals.

In terms of implementation both the supervisor and the viewer:

1) May act in a client-server manner. This means that they can split to a server submodule that resides close to the VRS (as a process running on the same machine) and another client submodule that runs anywhere in the Internet and acts as the front-end interface to the

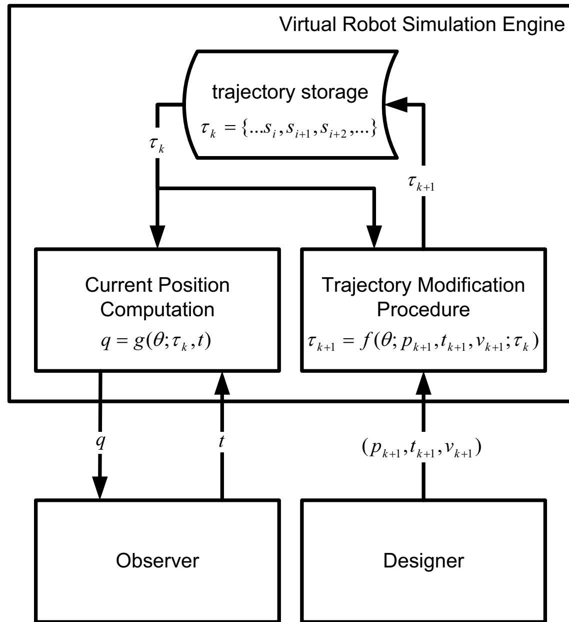

Fig. 2. The proposed architecture including the two types of elementary interface modules (Designer and Observer).

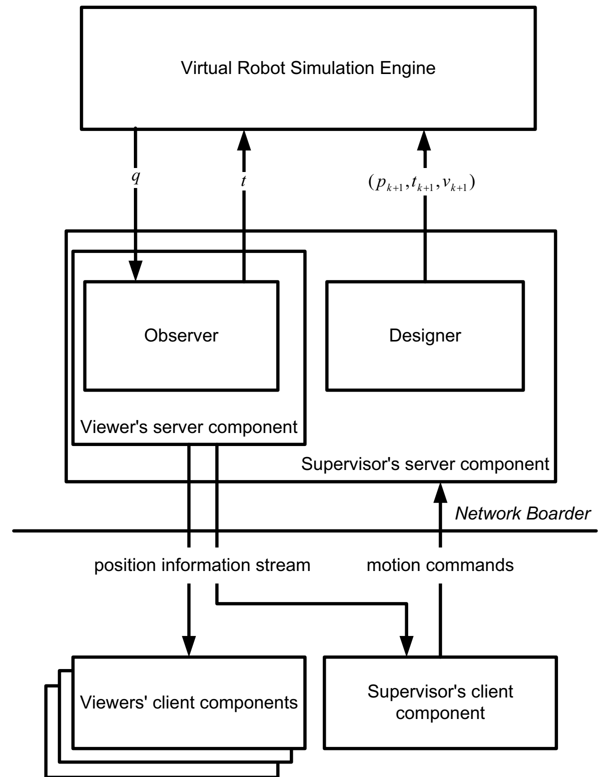

Fig. 3. Viewer and Supervisor Interfaces and the corresponding information flow. The Supervisor encapsulates the server side component of a viewer. Multiple Client-side Viewer components share the same server component. No time indication is being sent to the server, function g is periodically executed at a pre-agreed rate.

user. It is worth noticing that more than one viewers may operate simultaneously either by sharing the same server submodule or by spawning more such servlets. In particular, the server part of the viewer operates as a streaming server that delivers position information to the user in a fashion similar to video frames.

2) May be graphics enabled in the sense that position information and/or position commands (the latter for the case of supervisor only) are represented within some visualization environment (e.g., via VRML.)

It is also interesting to point out that in accordance to the proposed structure of the trajectory τ_k, viewers can be enabled with play-back capabilities (play, pause, rewind, etc.)

The architecture of the Viewers and the Supervisor is depicted in Figure 3.

IV. Operation over the Network

The client-server implementation of the supervisors and the viewers allows for remote interfacing to the VRS engine over any type of data networks (including the Internet).

A number of *network protocol* and *information coding* related issues should be resolved for achieving a stable implementation of such remote teleoperation. For the shake of brevity we skip this discussion in the context of this paper and concentrate only to the most important consequence of introducing the net in the loop: *delays*.

A trajectory modification command issued at time point t^{iss} reaches the VRS only after D_f seconds. Similarly the outcome of current position function $q = g()$ is delivered to the end user D_b seconds after the actual evaluation of the function (ref. Figure 4).

These delays are not important when the simulation engine operates in a *non* real-time mode; unfortunately, though, they may have serious consequences when a real time operation is required with the inherent constraints presented in Section II-C.

In particular, due to the forward delay D_f the first restriction of Section II-C should be transformed to $t^{exec} \geq t^{iss} + D_f$ and the second should assume $t_c = t^{iss} + D_f$.

A. Coping with the Delays

From a user point of view the forward (D_f) and backward (D_b) delays have the following annoying effects:

Motion commands should contain execution time-stamps (t^{exec}) referring to the future otherwise will be ignored or postponed. This means that *Supervisors (Designer component)* should act pro-actively.

Position parameters received by both the *Supervisors (Observer component)* and the *Viewers (Observers)* contain lagged (by D_b) information, i.e., they do not perceive the current robot position.

In the sequel we explore some methods to alleviate these effects.

The core idea is to use different "real-time" clocks for the server and for the client sides. The corresponding real-timelines will be synchronous but mutually shifted by a constant offset D. Under this assumption if t_u and t_e represent time index for user's and simulation engine clocks respectively, we assume that $t^e = t^u + D$.

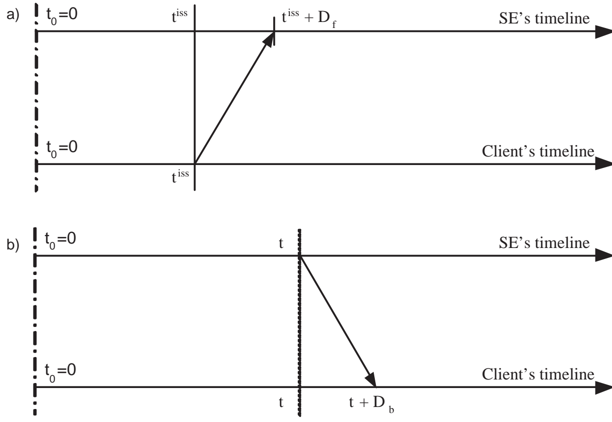

Fig. 4. The effect of (a) forward and (b) backward network delays.

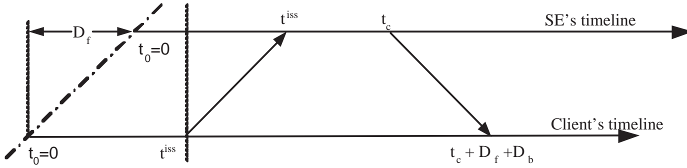

Fig. 5. The effect of shifting VRS time-line (t^e) by a *negative* offset equal to D_f w.r.t the time line of the users (t^u)

Setting $D = -D_f$ a motion command with $t^{iss,u}$ reaches the VRS at $t^{iss,u} + D_f$ i.e., when its local current time is $t^e_c = (t^{iss,u} + D_f) - D_f = t^{iss,u}$ (ref. Figure 5. Interpreting execution time-stamps t^{exec} using VRS local clock we cancel out the annoying effects of forward delays on the evaluation of the causality constraints. Essentially, this approach forces the VRS to operate in a delayed (by D_f) fashion w.r.t user's clock. Consequently, Viewers/Observers sharing the same clock with the Designers (like in the case of Supervisors) will perceive a total delay of $D_f + D_b$. This effect can be obviated if the server side of these modules is scheduled to "transmit" position information referring to future instances. In particular, at VRS local time t_c, execute $q = g(\theta; \tau_k, t_c + D_f + D_b)$ and transmit the q that is essentially a predicted value of robot's position. Of course this "trick" will result to invalid position information whenever prediction fails, i.e., when a motion command executed between t_c and $t_c + D_f + D_b$ happens to alter the trajectory.

On the other hand, setting $D = D_b$ (ref. Figure 6) the result of $q = g(\theta; \tau_k, t^e)$ evaluated and instantly posted at VRS local time t^e will be delivered to the user at $t^e + D_b$ which corresponds to user's local time $t^u = t^e - D_b = t^e$. Consequently viewers have the *illusion* that they obtain instant knowledge of robot's position.

V. Mixing Real and Virtual Robots

As already mentioned is section II-C the proposed model of virtual robot can be used as a useful stand-alone tool for teaching (including teleeducation in robotics), planning and experimentation (by allowing low cost simulation of real plants).

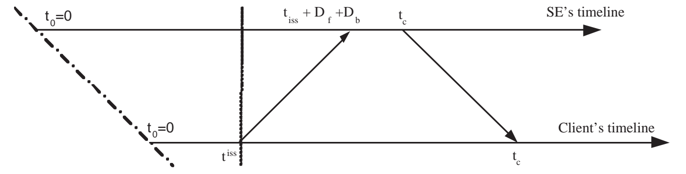

Fig. 6. The effect of shifting VRS time-line (t^e) by a *positive* offset equal to D_b w.r.t the time line of the users (t^u)

However, its most interesting use is when it acts as a representative of a real robot whose behavior is emulated by the virtual counterpart. In fact, settling a virtual copy of a real robot may yield a powerful visual feedback interface in telerobotic applications. Consider for example the scenario of steering a robotic arm residing in a remote location over a data channel with limited bandwidth and communication delays. Position commands of the form $(p_{k+1}, t_{k+1}, v_{k+1})$ are used for steering while functions f and g of the form in equations (1) and (4) represent the built-in kinematic behavior of the arm. Remote robot operator has not a direct visual feedback and conventionally this is substituted by installing a set of cameras on or around the robotic arm that capture and stream views of the robot over the available communication channel. Apart of the introduced delays and the resulting communication overload this approach does not allow a complete 3D view of the real robot.

A virtual emulator of the actual robot that (a) behaves in accordance to the very same functions f and g, (b) receives the same set of position commands, (c) is aware of the delays D_f and D_b characterizing the communication to the real robot, (d) outputs its position parameters to a virtual reality visual interface and (e)is conveniently installed very close to the operator yields a high quality feedback to its operator. The latter is in our terminology acting as a *Supervisor*.

VI. Implementation

A. Free design options

The methodology presented in Sections II through V provides a framework for implementing a virtual robot and various forms of interfaces that allow for issuing steering commands and acquiring real-time information regarding its position. In addition we have sketched the interesting combination of a virtual and a synchronized real robot. This framework is generic in the sense that it is actually neutral w.r.t.

1) the geometry of the robot(s)
2) the adopted kinematic model describing the detailed forward and inverse kinematic solutions used to describe the trajectory of the robots (both virtual and real) (i.e., the inner structure of functions f and g)
3) the syntax of the high level language used for posting commands and receiving position information
4) the exact format of the exchanged data (position parameters, time stamps)
5) the (software development) platform and the (graphical) user interfaces used to implement the VRS and the client-server components of the interfacing modules

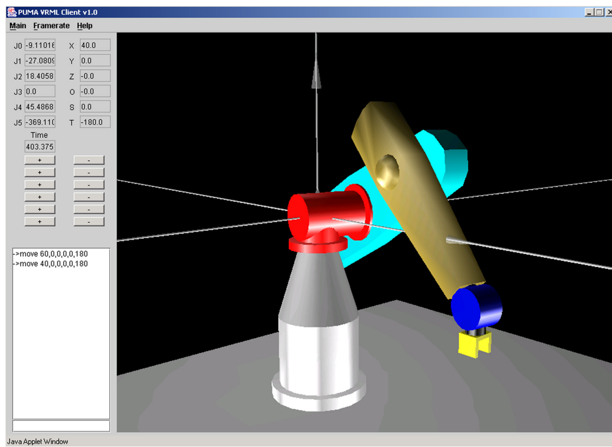

Fig. 7. The Supervisor interface of the virtual PUMA robot: The left panel is used to issue position commands (that will be forwarded to the trajectory designer module) and the right panel provides live visualization of the incoming position information stream.

6) the network protocols employed to support data interchange

Purposely following this abstraction we tried to decompose modelling from implementation issues.

B. Implementation choices

In our experiments we adopted certain options and made particular implementation choices that are described in the sequel.

The kinematic model adopted for the construction of functions f and g is fully compatible to a PUMA robotic arm with six rotary joints available in the Robotics Lab of our department. For compatibility reasons VAL-II has been used as a reference model for the high level syntax of motion commands and position specification.

The real time implementation of our VRS is based on Matlab v.6.5 release 13 and a number of forward and inverse kinematics routines emulating the kinematic model of PUMA.

The server side components of Viewers and Supervisors were developed in Java (version JDK 1.4.2 from Sun Microsystems). These components are essentially translating VAL-II position commands into Matlab calls that modify the trajectory structure as described in Section II-B. The client side interfaces are graphics enabled java applets that use VRML97 and Java3D to visualize the position of the virtual robot (ref. Figure 7).

Data exchange, i.e., the communication between clients and servers is based on TCP/IP.

Viewers, in particular, comprise of a broadcast server module shared by more than one clients. Each broadcast server transmits robot's position information periodically acquired via successive invocations of function g. More than one such servers can be spawned simultaneously delivering position "frames" at different rates to Viewer Clients that connect to them. Frame rates of up to 50 frames per second have been tested without serious overload of the Matlab based VRS.

VII. Conclusion

A generic model for simulating the kinematic behavior of robotic arms was presented in this work. The structure of a trajectory list consisting of time intervals where robot joints follow a piecewise polynomial path was used to represent robots time varying state. A formal method for updating this structure when time-stampted motion commands are issued was proposed. The corresponding algorithm was considered as the computational core of the proposed Virtual Robot Simulation (VRS) Engine. In addition, the same engine is equipped with the functionality of responding to position requests at arbitrary time instances. Constraints imposed by the operation of VRS in real-time mode were explored. Two elementary interfaces, namely the (trajectory) designer and the observer were presented and subsequently used to build user interfaces of higher level, i.e., the supervisor and the viewer. It was also described how the latter can be implemented in order to allow communication over a data network suffering from delays (such as the Internet). The use of the proposed simulation environment in conjunction with a real robot was proposed as a means to overcome the lack of direct visual feedback in teleoperation applications.

A brief report on our implementation of the proposed virtual robot modelling and the corresponding interfaces was presented after all.

The authors are currently exploring the many open issues not covered within the proposed scheme. Among them, the most interesting ones include (a) the handling of delay jitter (i.e., the time varying nature of network delays), (b) the efficient encoding of transmitted position information by considering tools like MPEG-4, (c) the extension of motion commands to continuous time steering mode (e.g., by dragging robot's tool), (d) the improvement of user interfaces by adding higher interactivity between the supervising user and the robot and (e) the improvement of the synchronization between the virtual and the real robot (by adding a direct mutual communication link).

References

[1] K. Taylor and B. Dalton. Internet robots: A new robotics niche. *IEEE Robotics and Automation Magazine*, 7(1), January 2000.
[2] H. Shen R. C. Luo, K. L. Su and K. H. Tsai. Networked intelligent robots through the Internet: Issues and opportunities. *Proceedings of the IEEE*, 91(3):371–382, 2003.
[3] Kuk-Hyun Han, Sinn Kim, Yong-Jae Kim, and Jong-Hwan Kim. Internet control architecture for Internet-based personal robot. *Autonomous Robots*, 10(2):135–147, 2001.
[4] I. Belousov, G. Clapworthy, and R. Chellali. Virtual reality tools for Internet robotics. In *IEEE Intl. Conf. on Robotics and Automation ICRA'2001*, pages 1878–1883, Seoul, Korea, May 2001.
[5] R. Oboe and P. Fiorini. A design and control environment for Internet-based telerobotics. *Intl. J. of Robotics Research*, 17(4):433–449, 1998.
[6] J. Tan and G. L. Clapworthy. Virtual environments for Internet based robots- i: Modeling a dynamic environment. *Proceedings of the IEEE*, 91(3):383–388, 2003.
[7] K. Fu, R. Gonzales, and C. Lee. *Robotics*. McGraw-Hill International Editions, 2nd edition, 1988.
[8] T. Yoshikawa. *Foundations of robotics*. Corona Publishing Co.Ltd, 1990.
[9] J. Craig. *Robotics Mechanics and Control*. Addison-Wesley Publishing Co., 1986.