Robotics education, teleprogramming, telecontrol through the internet

Giuseppina G. Gini, Politecnico di Milano, DEI, Milan, Italy

Abstract—**Teleoperation allows a human operator to use a visual display and a joystick to manually control a remote robot. The wide spread of internet is pushing now toward the use of internet instead of a dedicated connection, so more emphasis is now given to the topics of programming in internet and of using virtual reality. We investigate here the concept of teleprogramming and illustrate a system for a 6dof robot arm. In particular teleprogramming solves the delay problem of using a remote robot by giving more autonomy to the robot and sending it only high level commands.**

I. Introduction

A well-established and growing need exists for the capability to perform work remotely using teleoperation and telerobotic systems. Teleoperation allows a human operator to use a visual display and a "master" manipulator (e.g. a joystick, or a mouse) to manually control a remote "slave" device such as a robot. Presenting realistic visual information and contact forces to the operator improves task performance and increases the sense of telepresence [1].

Telerobotics combines the concepts of teleoperation with robotics and automation, enabling the human operator to supervise the execution of a remote task rather than exercising continuous manual control.

Under supervisory control [1] [2], a slave with autonomous capabilities can be given discrete, high-level instructions rather than continuous commands, thus relegating the human operator to the role of supervisor. Such instructions must first be programmed or taught to the slave, and the operator must continually monitor and update instructions during execution. This method is therefore only useful when the automation is trustworthy, the task execution time is larger than the delay time, and the unpredictable aspects of the remote environment are changing very slowly.

The technique developed here is to predict the real-time behaviour based on modelling the slave manipulator and environment in virtual reality, and to provide this as a feedback to the user. This approach is embodied in a control method called teleprogramming and used also to give a feedback to the real robot; here we develop the robot program generation, where high-level instructions are generated automatically based upon the operator's actions, and the slave manipulator autonomously executes these commands.

Through ViMa (Virtual Manipulation), we built a teleprogramming system for a CRS 6dof robot, including pointing interface, collision avoidance, and multiuser access. The system is implemented in C, Java, and VRML. Teleprogramming simulates direct teleoperation by allowing the user to interact with a graphical simulation of the remote environment and manipulator. High-level instructions are generated automatically based upon the operator's actions, translated into robot instructions, and executed.

Moreover ViMa is an example of the virtual laboratory, a concept important in education and supported by international agencies. The virtual robot laboratory is a copy of a real laboratory. In this manner it is suitable to deliver the know-how for the presentation of virtual robot scenes, for virtual robot programming as well as real applications, and for processes where robots are involved. The real robot laboratory is able to be controlled with the telematic methods and thus the laboratory may be used as well as global and as also local laboratory.

For the off line robot programming we need an excellent virtualisation of the robot working place and of the whole behaviour of the robot. It is necessary that the scene can be viewed from different points and from changing directions. An excellent virtualisation is absolutely necessary when we program the robots off line, in some cases without seeing the real robot.

The use of the virtualisation techniques for robot scenes is very useful both for the robot programmers, and for the robot program beginners and for university students.

For example, knowledge in kinematics, coordinate systems and transformations, is better assessed using examples and

This work was supported in part by the contract ENEA-MURST, project SIRO.

understanding them. We do not need a real robot for that, but interactively moving robots on a screen may accelerate the learning process.

In the following we will illustrate the design criteria and the results of our system. Section 2 gives a general overview of the architecture. The following sections are dedicated to the server, which is responsible for most of the modelling and control strategies: section 3 is about the kinematics problems of our system, section 4 illustrates the main algorithms to compute moves in the geometric world before sending them to the robot.

II. Overview of ViMa – Virtual Manipulator system

Our system implements teleprogramming through a client-server architecture and the TCP-IP protocol. As said before, the model of the world is used by the programmer to obtain both visualization of the scene and the programming interface to the robot. The complete architecture of the system is illustrated in Fig. 1.

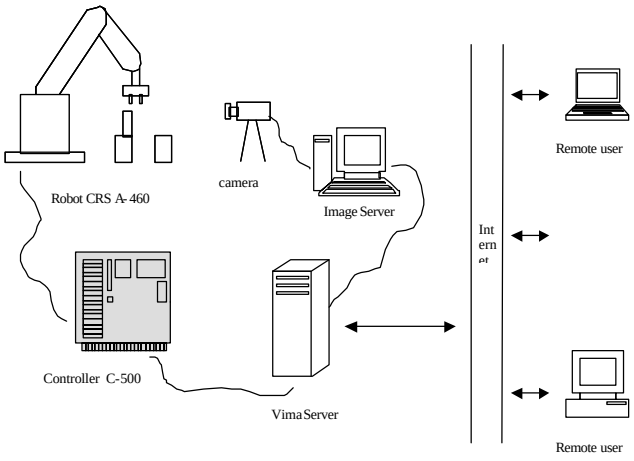

Fig 1. The structure of the ViMa system, with the main components on the server side, and the connections through internet.

A. ViMa Client

The client is a Java applet, usable in any browser; the virtual world is represented in VRML (virtual reality modelling language) [3] and managed through the EAI (external Authoring Interface) [4].

The functions of the client, illustrated in Fig. 2, are:

- Managing the virtual world (creation, insertion and deletion of objects, saving and loading);
- Moving the robot.

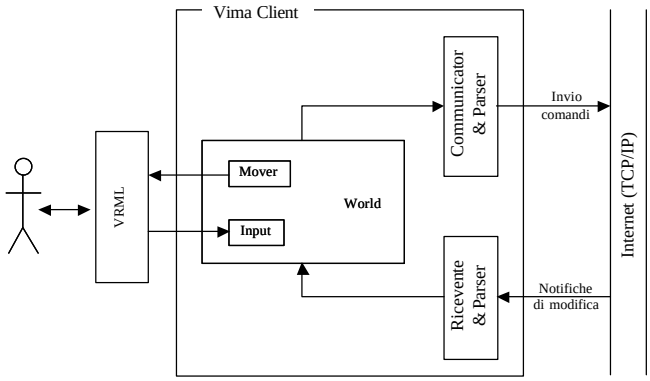

Fig 2. The structure of the ViMa client.

B. ViMa Server

The ViMa server is a C++ application for Windows The user interface on the server, illustrated in Fig. 3., shows in a text area all the operations asked by the client and sent to the robot controller

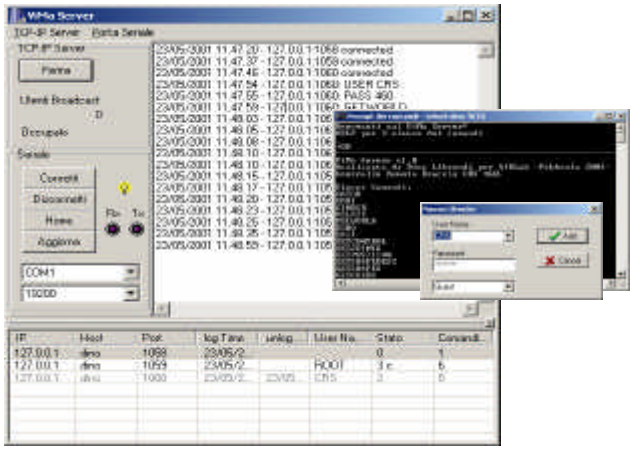

Fig. 3. The user interface on the server.

More clients, so more users, can concurrently access the server. To maintain the compatibility of the worlds seen by any user, the server upgrades the virtual world according to the robot actions taken, and propagates the modifications to all the clients, so all the users can continue to work on the real situation.

The server is responsible of executing many functions, the main being:

- Computing the direct and inverse kinematics;
- Communicating with the robot controller;
- Receiving data from the clients (also commands for the

robot);

- Sending data to the clients (world modifications);
- Trajectory computation;
- Collision detection.

C. *The robot CRS A-460*

The robot used in our system is a CRS A-460 [5], an articulated arm with 6 dof., illustrated in Fig.4.

Fig. 4. The real CRS robot.

The controller of the robot is the C500 controller [6], hosted on an Intel 8086 processor. In the BIOS of the controller there is a simple multitask operating system, and the programming language is RAPL-II [7], a Basic-like language. The controller is connected through Rs-232 at 19200 baud to a computer where the programs are written. In our application, the trajectories are computed by the server and translated into RAPL code, sent to the controller. A class is devoted to the robot communication.

The VRML model of the robot arm has been developed and calibrated. The temporal specifications needed to execute programs with given trajectories are managed through timers.

D. *Using ViMa Client*

1) *Initialisation*

The initialisation requires to physically connect the robot and the controller, the controller with the server PC server (through the RS-232 door of the controller and the serial of the PC). After putting On the controller of the arm, we can start the ViMa Server (select the serial door and set 19200 bps, start the TCP-IP Server). To start a client, we suggest the Cortona VRML Client or the Cosmo Player 2.x. The starting window is in Fig. 5.

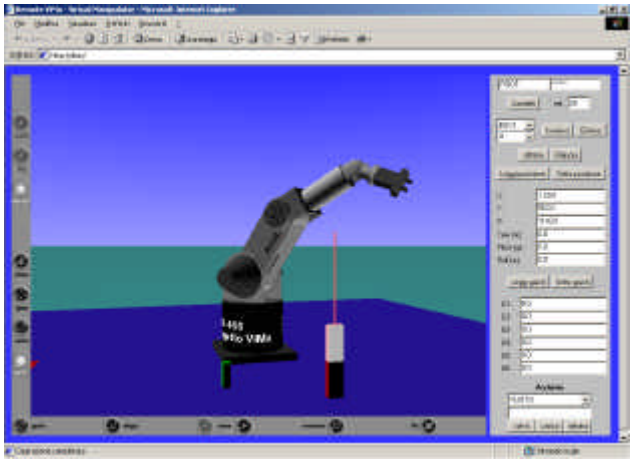

Fig. 5. Starting the client: the virtual CRS robot.

2) *Client operations*

Different categories as guest, user, superuser, are defined. Any category can move the robot. The ViMa client can send commands in 4 ways::

- Inserting the 6 joint values.
- Inserting the 6 Cartesian values;
- Using the task commands (“grasp”or “release”;
- Interacting with the objects in the VRML world, and the action is performed by the robot.

The Java applet makes:

- Connection to ViMa server;
- Insertion or deletion of objects in the virtual world.
- Pick & place.
- Direct and inverse kinematics (computed on the server).
- Filing options..

III. KINEMATICS STUDY: CALIBRATING THE MODEL

The conversion between Cartesian and joint coordinates has been developed for our robot, considering that the CRS is a closed system and no information are available about the internal algorithms..

The robot model was built in the Denavit-Hartenberg notation, and published in [8].

It is necessary to calibrate the robot arm in the real world taking specific calibration positions, sending there the robot and modifying the parameters to reduce the difference between the real and robot coordinates. Since we use the model of a plane working area, the possible errors are due to an offset, to scale errors in the axes, to errors in the orientation of the plane with respect to the robot reference system. The commanded position [*X, Y, Z*] is a linear combination of the components of

the position we want to reach [*x, y, z*]:

$$\begin{cases} X = a\,x + b\,y + c\,z + k_x \\ Y = d\,x + e\,y + f\,z + k_y \\ Z = g\,x + h\,y + i\,z + k_z \end{cases} \quad (1)$$

To obtain the value of the 12 parameters we need at least 4 trials. Two methods are used:

1) Exact method

If the 4 measures have no errors we can solve the system and obtain the values. Let us consider the case of determining *X*. We write a system of 4equations with 4 unknown, obtained from 4 measures: X_i are the commanded values and x_i, y_i, z_i are the measured values:

As a matrix equation we get:

$$\begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ x_4 & y_4 & z_4 & 1 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \\ k_x \end{bmatrix} \quad (2)$$

$$\mathbf{x} = \mathbf{M} \cdot \mathbf{v}$$

To obtain **v**, the parameters, we invert **M**

$$\mathbf{v} = \mathbf{M}^{-1}\mathbf{X} \quad (3)$$

This method however is too sensible to measures errors.

2) Mean squares estimate

A more robust method tries to minimize the error function.

Let $\mathring{a} = X - (a\,x + b\,y + c\,z + k_x)$ be the estimated error given a, b, c, k_x. We want to find the values of the parameters that make minimum $\mathring{a}$ We can use the sum of the squares. For *n* points, $f(a,b,c,k_x) = \sum_{i=1}^{n} e_i^{\,2}$ is the function to minimize. We compute the square and when the derivative is null:

$$e_i^{\,2} = X_i - (a\,x_i + b\,y_i + c\,z_i + k_x)^2 =$$

$$X_i^{\,2} + (a\,x_i + b\,y_i + c\,z_i + k_x)^2 - 2X_i(a\,x_i + b\,y_i + c\,z_i + k_x)$$

$$\frac{\partial \sum e_i^2}{\partial a} = \sum \left[2(a\,x_i + b\,y_i + c\,z_i)x_i - 2x_i X_i\right] = 0 \quad (4)$$

For the other variables, in matrix form:

$$\begin{bmatrix} \sum X_i x_i \\ \sum X_i y_i \\ \sum X_i z_i \\ \sum X_i \end{bmatrix} = \begin{bmatrix} \sum x_i^2 & \sum x_i y_i & \sum x_i z_i & N \\ \sum x_i y_i & \sum y_i^2 & \sum y_i z_i & N \\ \sum x_i z_i & \sum y_i z_i & \sum z_i^2 & N \\ \sum x_i & \sum y_i & \sum z_i & N \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \\ k_x \end{bmatrix} \quad (5)$$

We solve as before, after inversion, and we get a more robust estimate.

IV. The Role of the Virtual Model

The user can modify the virtual world by adding new objects, removing objects, grasping or releasing, move without collisions. The algorithms are called using a pointing interface which interprets the movements of the cursor. Since the real world is 3D but the cursor is in 2D the algorithms compute the third coordinate using heuristic. Trajectories are generated using the via points heuristically generated and computing the collision; if a collision is detected the trajectory is modified.

Most of the robot operations require grasping. In the pointing interface grasping is obtained checking the presence of a pre-defined grasping position for the object, computing the approach point, and modifying the world model accordingly. The release of an object is done after checking that the release position is suitable for a stable pose. All the algorithms use heuristics to get the precision needed by the robot, which has a declared repeatability of 0.05 mm., so a poorer accuracy.

1) Inserting and removing objects

A problem in moving objects is to check the stability conditions of the structures. The conditions to check to have a minimal stability of the object when released are about the presence of a support (other object or plane) under to object before releasing it. The simplest point to check is the centre of mass of the object. The vertical position is found using a dichotomise research and moving on the Zaxis direction. After n steps the position is found with an approximation $e < \frac{l_z}{2^{n+1}}$, with l_n dimension of the object. In case l_z=50mm and n=2*5, the positioning error in the model is 0.02mm

To delete an object, the program checks the existence, the position to eliminate the risk of removing an object in the hand or below another object; then it modifies the list of the objects. See Fig. 6 for insertion.

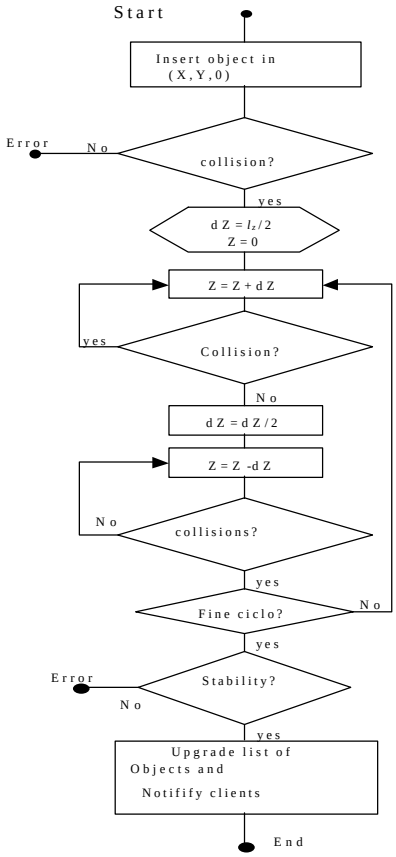

Fig6. Insertion flowchart

2) Setting Cartesian and joint positions

The Cartesian procedure calls the inverse kinematics to compute the joint positions. The joint assignment works directly. Moreover, the joint assignment calls the direct kinematics, updates the positions of the grasped objects, and verifies collisions. The flowchart is in Figure 7.

3) Generate a trajectory without collisions

The trajectory is generated from start to goal checking for collisions. See Fig. 8.

The obstacle avoiding with a 6 degrees of freedom robot is quite complex, and in the literature important results show that is almost intractable in C-Space.

The library SOLID (Solid Interference Detection Library) to detect collisions in a virtual world [9, 10], is a C++ STL library to measure the distance of convex polygons. It employs the Qhull algorithm [11] to determine the convex hull and to generate the adjacency graph. The algorithms extracted from the SOLID library have been used in Borland C++Builder, and modified in some termination conditions.

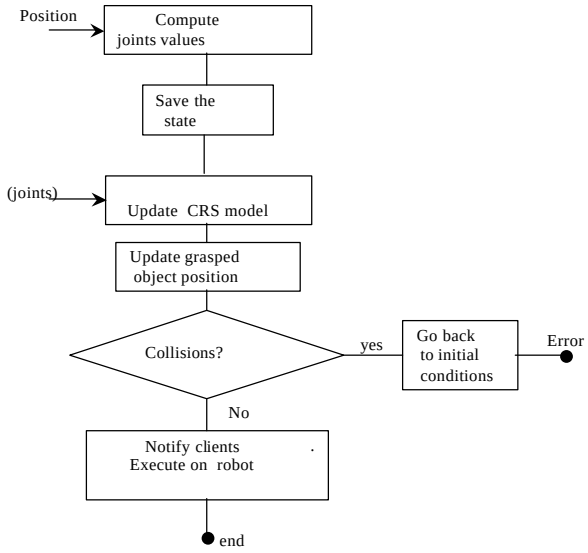

Fig 7. Cartesian and joint position command

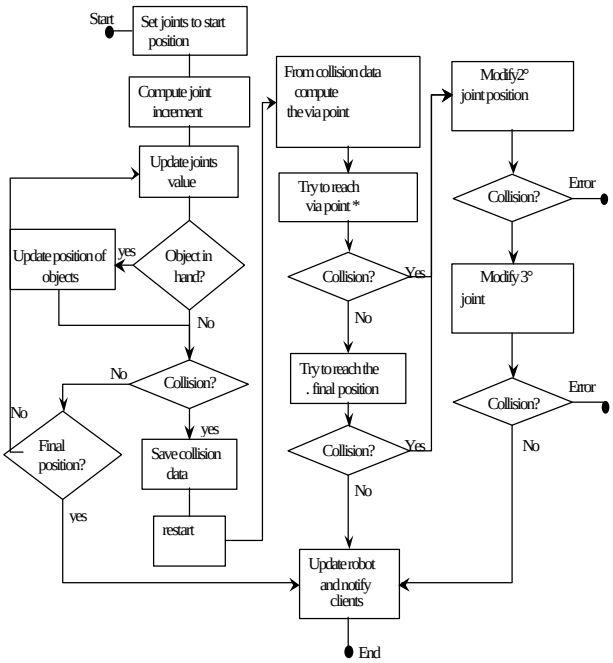

Fig. 8. – Compute a trajectory flowchart

4) Catch and release of objects

We have a grasp position stored with the object models. The catch operation requires to individuate the object and its position, to compute the approach and the path (to avoid obstacles), to grasp the object, to go to the final position, to updare the world, as illustrated in Figure 9. The release operation is the reverse: determination of the release position,

checking the stability of the position, then execution.

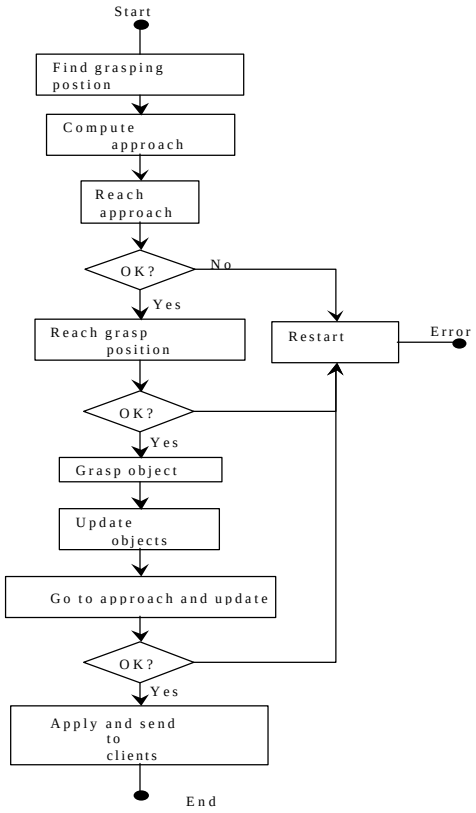

Fig 9. Catching an object flowchart

V. CONCLUSION

The main point of ViMa is that it is straightforward to use it by any user. It is possible simply to translate the commands given by the pointing interface into movements of the robot. Facilities as collision detection and avoidance are able to make successful the obtained robot program.

The client is portable and small, so any user can use it on simple computers. The geometric and kinematics computations are made on the server, with a good precision.

We have verified the critical points, where numerical[1]. problems are possible. All the cycles are controlled by a Watch-dog[2]), so the server closes the connection and the thread and works .on another client.

The algorithms running on the server, as the collision detection and avoidance, are computationally heavy. On a PC-INTEL 600MHz, the transportation of an object (requiring many collision tests) with 50 objects in the scene requires 0.5-1.5s. The algorithms running on the client employ coordinate transformations and make extensive use of visualization. If a video board is available, the performances are good. We have 1-5 frames per second on a basic PC (with render software), and 5-20 fps for old video boards, to 40-50 fps for new video boards (render OpenGL, DirectX).

The client can be slow in receiving the upgrading from the server (communications delays). The need of a camera to show in a window of the screen the real world is clear, and a Java application for a camera mounted on a pan&tilt unit is ready. This solution is not integrated into ViMa because the vision algorithms are too heavy to be hosted on the same server as ViMa.

Other systems are reported in the literature to support teleoperation of a robot manipulator. In [12-14] we see other solutions for manipulation and even access the most known telerobotic sites. However our approach is the only one using a task level language in the user interface.

[1] Numerical problems: overflow, underflow, /0, null in.arcotangent.

[2] A timer is reset after a control loop. In case of errors the timer generates an interrupt and restarts.

REFERENCES

[1] T. Sheridan, “Human Supervisory Control of Robot System”, *Proc. IEEE Conference, International Conference of Robotics and Automation*, San Francisco, Apr 7-10, 1986

[2] T. Sheridan, “*Telerobotics, Automation and Human Supervisory Control*”. MIT Press, Cambridge, Massachusetts, 1992.

[3] G. Bell, R. Carey, C. Marrin. “*VRML97: The Virtual Reality Model Language*”, available in http://www.vrml.org/Specifications/VRML97.

[4] C. Marrin, ”Proposal For a VRML 2.0 Informative Annex: External Authoring Interface Reference”, Silicon Graphics

[5] A460 Series Small Industrial Robot System, Technical Manual, CRS Plus Inc., Canada, 1992.

[6] C500 Series Small Industrial Robot System Controller. C500 Operation Manual, CRS Plus Inc., Canada, 1995.

[7] *RAPL-II Programming Manual*, CRS Plus Inc., Canada.

[8] G. Gini, D.Librandi, “Teleprogramming a robot arm through the internet and virtual reality ”, Conferenza Robotica, Enea, Frascati, Nov 2002, p 185-192..

[9] . http://www.cs.unc.edu/~geom/I_COLLIDE.html

[10] M. Lin, “*Efficient Collision Detection for Animation and Robotics*”. PhD Thesis, Computer Science, University of California, Berkley, 1993.

[11] Barber, Dobkin, Huhdanpaa “The Quickhull Algorithm for Convex Hulls”. *ACM Transactions on Mathematical Software*, Vol 22, N. 4, pag. 469-483, 1996.

[12] J. Lloyd, J. Beis, D. Pai, D. Lowe, “Programming Contact Task Using a Reality-Based Virtual Environment Integrated with Vision”, *IEEE Trans R&A*, June 1999 pp 423-434, 1999.

[13] http://www.robotic.dlr.de/VRML/Rotex/index.html

[14] . http://telerobot.mech.uwa.edu.au