DEVELOPMENT OF AN OFF-LINE PROGRAMMING (OLP) SYSTEM FOR A SERIAL LINK ROBOT MANIPULATOR

Faisal Zubair Qureshi, Muhammad Asif, Mahmood Ahmad and Abdul Rauf
Informatics Complex, P. O. Box 2191, H-8, Islamabad

ABSTRACT

For robot manipulators, an *OLP system* provides a detailed 3D simulation test-bed for visualization and analysis of various what-if scenarios. This paper discusses the prototype OLP system developed at the Informatics Complex for a six degrees of freedom cylindrical serial manipulator. Paradigms from classical robotics literature have been used to develop models of kinematics and to plan manipulator trajectories. 3D graphics techniques have been used in an object-oriented programming environment; Microsoft Windows was chosen as the platform for the OLP system. The simulator provides a complete and flexible view of the manipulator, with menu driven task simulation capability, and tools for visual and mathematical analysis of manipulator's performance. Mathematical models for kinematics and trajectory planning for the manipulator have effectively been integrated with the developed 3D simulation software.

1. INTRODUCTION

A program for indigenous development of robot manipulators was started at the Informatics Complex, Islamabad, a couple of years ago. Most of the projects initiated then [1-5] are now in the final fabrication stage. The program included development of serial link manipulators. Serial manipulators have links that are connected with each other serially through movable joints. A properly designed serial link manipulator attempts to replicate the working of a human hand. Typically, the first three links form the synthetic arm and the last three links emulate a wrist. These are used in environments where, due to reasons of radiation hazards, excessive mechanical loads or demands of high precision, etc., employing a human is not desirable.

A practical robot manipulator consists of a mechanical structure assembled according to a detailed design. Its software contains models of kinematics and dynamics, algorithms to plan trajectories and modules to control manipulator motion. The hardware consists of sensors to interact with the environment and circuitry to implement the control logic. A robot manipulator is thus a complex system and its performance in executing a given task is often difficult to predict. This phenomenon, therefore, becomes the prime motivation for developing an OLP software for robot manipulators. An OLP system provides a software environment where tasks to be performed by a manipulator can be simulated in detail. An *ideal OLP system* will have the following features: the computer screen will be used to create a 3D environment *exactly* representing the actual manipulator and its surroundings. The mathematical models of kinematics, dynamics and trajectory planning will be incorporated exactly as the way they are in the software of the actual manipulator. It will also contain details of sensor processing and control algorithms, etc. However, as is typical in developing any simulator, *the practical OLP system* will accommodate the trade-offs between accuracy, processing time and financial constraints of sophisticated hardware. A 3D environment is essential because robot simulation with only 2D effects will always be incomplete. The robot industry has continued to propose the use of an OLP system to supplement the performance of a manipulator. A number of vendors have come up with OLP systems for their particular commercial robots. Silma Inc.'s SimStation [6] and IBM's AUTOPASS [7] are a few examples. The need for indigenous development of an OLP system was felt because of its high cost in the international market. The prototype OLP system developed at the Informatics Complex has been named ***InfoSim.*** It provides simulation environment for InfoMate, a six degrees of freedom (DOF) cylindrical robot manipulator, currently under fabrication at the Scientific & Engineering Services, Islamabad.

Development of InfoSim required detailed information about the physical structure of InfoMate and the corresponding mathematical models; it also required clever programming to come up with a powerful real time 3D interface. The best available machine was a GATEWAY2000 P-90. The paper describes salient features of InfoSim and the issues involved in the process of its development.

2. SALIENT FEATURES OF INFOSIM

As discussed in the introduction, InfoSim was developed for a serial manipulator named InfoMate. Figure 1 provides a 3D view of the InfoMate. The manipulator without the wrist has one revolute and two prismatic joints. The revolute joint provides rotatory motion about the central vertical shaft, whereas two prismatic joints provide linear motion along and across it. The wrist design consists of three revolute joints. This architecture provides six degrees of freedom (DOF) to the manipulator.

InfoSim was developed for the MS Windows 95 platform using Visual C++ 4.0. The choice of the platform and the software environment, the problems encountered in developing the OLP software and the techniques used to overcome these problems are discussed in the next section. The manipulator model as seen on the simulator screen is an *exact* 3D representation of the actual InfoMate, i.e., the software representation was developed according to the fabrication drawings of the manipulator. InfoSim provides a robust synthetic camera; the user thus gets a chance to visualize the

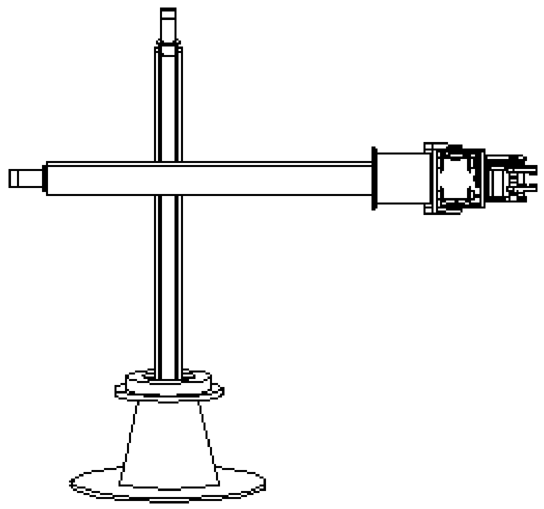

Figure 1: InfoMate – a six degrees of freedom cylindrical robot manipulator

manipulator from any position and orientation in the simulation environment. The interface also allows the user to provide arbitrary set of manipulator joint values and updates the display accordingly. InfoSim provides the option to visualize the manipulator end-effector and the end-effector frame. The option to see the global frame of reference superimposed on the end-effector frame is also available. Thus the orientation of the end-effector frame with respect to the global frame can also be visualized.

For InfoMate, models of position, velocity and acceleration kinematics were developed and rigorously tested. Forward kinematics paradigms are used to calculate position & orientation, linear & angular velocities and accelerations of the end effector from the available joint information. Inverse kinematics involves extracting joint information from the available end-effector position/motion information [6]. InfoSim contains *software engines* incorporating these models. The end-effector position is represented as the x, y and z co-ordinates of the global frame of reference. Orientation is available as a 3x3 orientation matrix and as the three ZYX fixed angle values.

A typical manipulator task is to move the wrist (end-effector) from a certain position & orientation to a desired position & orientation, usually to move/pick an object of interest. InfoSim provides facilities for manipulator task simulation. The software environment allows the user to place a cuboid object of user specified dimensions at any position & orientation. As discussed above, the forward position kinematics engine provides information about the current position and orientation of the manipulator. The interface asks the user to provide position and orientation of the wrist at the destination. Task simulation in InfoSim is to move the manipulator from its current position and grab the object at destination. One option is to manually enter the destination x, y and z coordinates and the nine entries of the orientation matrix. Manually constructing an orientation matrix for a certain orientation is cumbersome; InfoSim provides the facility to visually specify the desired orientation (and position) of the wrist at destination. A toolbar is available to visualize the tip of the wrist at/around the object and to incrementally change the orientation about any fixed axis resulting in a corresponding update of the orientation matrix.

The next step is to specify a trajectory scheme. If the user is only concerned with destination position & orientation and the intermediate configurations of the manipulator are not of any interest, then *joint space trajectory schemes* are suitable. Polynomial functions of time are *fitted* such that the initial and final joint velocities are zero, thus providing smooth, jerk free manipulator movement. InfoSim provides the choice between cubic polynomials and linear segments with parabolic blends (LSPBs). Option to choose between simultaneous or sequential joint motion is also available. The inverse position kinematics engine provides joint value information corresponding to the destination position & orientation. In case of a six degrees of freedom cylindrical manipulator, the solution to the inverse position kinematics problem is not unique. Two sets of joint values correspond to an end-effector position and orientation. InfoSim allows the user to choose between the two solutions. If the end-effector position and orientation during the intermediate stages of the motion are of interest, then *Cartesian space trajectory planning* is used. Whereas theoretically, the end-effector can be made to follow any path in the three dimensional space, InfoSim provides only a linear Cartesian path option. Problems with Cartesian paths are abundantly discussed in robotics literature [6,8]. For a given Cartesian path, InfoSim reports presence of work-space violations, singularities and excessive joint velocities & accelerations, if any. It offers choice to closely visualize the manipulator motion in the 3D simulation environment by providing instantaneous forward & reverse replay and a visual representation of the end-effector trajectory. Options to observe graphs of joint motion and end-effector motion are also available.

Simulations of manipulator colliding with obstacles in the environment are also available. Collision detection can be done visually.

3. OLP SYSTEM – SOFTWARE ISSUES

3D modeling of physical systems is a demanding job. It requires computational power, fast hardware and a careful choice of algorithms. The first OLP system prototype was a DOS based simulator incorporating kinematics and 3D model of only three DOF cylindrical manipulator. The system was slow and beset with flickering and memory problems. Hidden-face elimination was done using *painter's algorithm*. This algorithm requires lesser memory but needs to sort polygons according to their depth. This is time consuming for complex objects as the time elapsed is proportional to the square of number of polygons [9]. If two polygons are intersecting, then mere sorting does not help. As a solution, one or more polygons have to be sub-divided before sorting is performed. The other choice was *z-buffer technique*. In this technique, each pixel on the view plane is assigned color of the surface with the smallest *z*-value (depth) at that pixel position. A depth buffer is used to store *z*-values for each position as surfaces are compared [10]. This algorithm seemed to be better for InfoSim since it is easy to implement and the processing time does not increase with the complexity of the scene. The depth buffer, however, usually runs into megabytes and DOS does not easily allow memory access beyond the 640K limit. Managing a buffer greater than this size is a challenging task in software development. Because of memory requirements, *z*-buffer was difficult to implement in DOS. This was one of the reasons, in addition to flickering and low resolution problems, that DOS was rejected. Moreover, DOS is fast becoming obsolete. Microsoft Windows provides a better user-interface. The lure of a dazzling array of graphic interface elements and a host of other Application Programming Interfaces (APIs) was not enough to choose Windows as the platform for the simulator since in a 3D simulation, no body cares much about pretty dialogue boxes. The name of the game is fast screen updates. Two things have come together to cure Windows inability to handle high-speed screen updating of the kind that 3D simulation software requires. One is the advent of Device Independent Bitmap (DIB), which allows developers to use their own customized code to render into off-screen buffers. The second is WinG, which creates the interface whereby DIBs can be moved to the screen at blazing speeds. Without these two features it would not have been possible to develop the software for Windows. In addition to that, memory management comes for free. Consequently Windows was chosen as the platform for InfoSim.

Visual C++ environment was chosen since it combines the benefits of Object-Oriented Programming (OOP) with powerful windows programming tools. Special care was taken to make the code as reusable as possible. Nearly 60% of the code is general purpose and can be used in other OLP systems. Document/View architecture of Visual C++ has further simplified the programming task.

Figure 2 shows different modules/sub-modules used in the OLP system and gives an idea about how they interact with each other. At the topmost level, the application can be divided into three segments namely, *user-interface module*, *3D graphics engine* and *robot modeling module*. Subsequent paragraphs briefly discuss these three modules.

3.1. Robot Modeling Module

This module contains models for kinematics and trajectory planning. Algebraic expressions for position, velocity and acceleration kinematics were optimized using Mathematica^{TM} [11], a state of the art symbolic computing package. This resulted in a significant improvement in the processing time. On a 40 MHz. 386 IBM PC-compatible, the processing time for forward position and velocity kinematics improved from 78.6ms and 159ms to 25ms and 18ms respectively, resulting in 68% and 88% improvement in the execution speed of the forward position & velocity kinematics engines. The inverse position kinematics problem was also efficiently solved using Mathematica. A number of C++ classes were developed to implement the functionality of this module. *InfoMotion* is the base class. As the name implies, this class encapsulates the functionality which brings about motion in the virtual manipulator's body. These motions are defined in terms of transformations on vertices defining robot's body and other related elements. Two types of transformations are used for this purpose: translation and rotation. Once transformations are applied, the module updates the *current status buffer*. The 3D graphics system peeks in to the *current status buffer* and draws *virtual* InfoMate on screen accordingly. Thus, *robot modeling module* and *3D graphics engine* are isolated from each other. *InfoMath* class inherits *InfoMotion* publicly. Methods are available in this class to implement the forward and inverse kinematics modules. This class also contains instances of classes *InfoJointTraj* and *InfoCarTraj*. Class *InfoJointTraj* has methods to compute joint space trajectory; similarly Cartesian space trajectories are calculated by *InfoCarTraj*. Both these classes have methods for generating data for graphs of different parameters. Class *FGraph* uses this data to display graphs on screen employing data normalization and validation techniques.

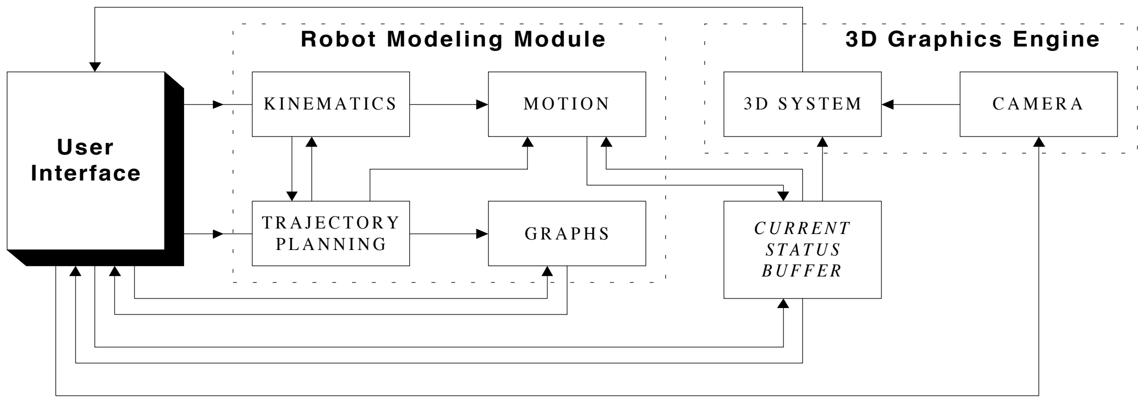

Figure 2: Different Modules used in the OLP System

3.2. 3D Graphics Engine

This module is responsible for the 3D representation of the *virtual* robot on the computer screen. Whenever contents of *current status buffer* are modified, *3D graphics engine* is asked to update the picture on the screen. The technique of *Polygon Graphics* was used to model the objects. In this technique, every object is approximated using polygons. Each polygon has some vertices and a color, e.g., a cube can be modeled using 6 polygons where each polygon has 4 vertices. To display the object on the computer screen each vertex is transformed using 3D to 2D mapping. To accurately generate a 3D scene on a computer screen, *hidden face elimination* and *back face culling* were incorporated. Class *FWorld* has all the functionality required to generate a 3D scene. It contains a list of objects' polyhedra and an instance of class *FCamera* –providing the functionality of a synthetic camera. It also contains an instance of class *FWingWin,* which fuses together the custom rendering code, Visual C++ and WinG technology. Class *FWorld* also manages a depth buffer used for *z*-buffer technique. This class is inherited publicly by the class *InfoWorld* that introduces additional features to the 3D-system, more specific to manipulator's environment like a grid, frames, trajectory and end-effector point. These are not real life objects, hence they were not incorporated in the class *FWorld* which is more general purpose. Any object seen on the screen is in fact an instance of class *FPolyObj.* A number of methods are available in this class to provide a standard way to define 3D objects. Class *FWorld* basically maintains a list of objects of class *FPolyObj.* Supplement routines have also been developed to generate the code for defining objects having user-specified parameters. A major portion of the code written belongs to the *3D graphics engine*. The library *Fas3dSim* was also created in which different drawing primitives like polygon rendering & line rendering are available; WinG technology was implemented.

3.3. User-Interface Module

Windows applications are famous for their standard graphical user interface (GUI). Since this is also a Windows application, it supports nearly all the elements of the GUI provided by windows like *menus, toolbars, dialog boxes* and *message boxes,* etc. It provides the user with an intuitive, attractive and easy-to-use interface through which user can simulate different tasks of the manipulator. *InfoSim* is an SDI application created through AppWizard which generates the basic skeleton of the application; MFC 4.0 library for Visual C++ was used.

4. CONCLUSIONS & FUTURE WORK

A prototype OLP system for detailed 3D simulation of a serial link robot manipulator has been discussed in this paper. As mentioned earlier, the developed software incorporates only the models of manipulator kinematics and trajectory planning. Work is underway to include models of manipulator dynamics, controls and sensor processing in the OLP system. Also, instead of the elementary visual collision detection currently available, algorithms for *automatic collision detection* need to be implemented. However, as is expected, adding further computational burden will tend to adversely affect the performance of the simulator. In these circumstances, use of dedicated graphics hardware will provide the necessary boost. Once the graphics hardware is available, adding finer points of 3D graphics like shading, shadowing and clipping will become expedient. The 3D interface can also be used to provide an on-line visualization capability when the manipulator is desired to be used in a master-slave configuration for remote operations.

5. REFERENCES

[1] Abdul R. Khan, Ishtiaq R. Khan, and M. Ather Syed, "Modeling and Simulation of Two Legged Walking Robot," *Proc. of the 1st IEEE (Pakistan Sec.), National Multi Topic Conference,* Rawalpindi, Nov. 1995.

[2] M.A. Syed, Farhan-Ullah, and M.S. Koul, "Modeling and Simulation of a Stewart Mechanism," *Journal of the Institution of Electrical and Electronics Engineers Pakistan, Vol. XXXIII,* July-December 1995.

[3] M. Ather Syed, Aslam Pervaiz, M. Saleem Koul, Farhan Ullah, Shaista B. Naqvi, and M. Anwar, "Intelligence Guidance and Obstacle Avoidance of an Autonomous Mobile Robot," *Proc. of the 1st IEEE (Pakistan Sec.), National Multi Topic Conference,* Rawalpindi, Nov. 1995.

[4] Naseem A. Khan, Amir S. Khan, and M. Ather Syed, "Neuro- Fuzzy Model for Reactive Behavior Control of an Autonomous Mobile Robot," *Proc. of the 1st IEEE (Pakistan Sec.), National Multi Topic Conference,* Rawalpindi, Nov. 1995.

[5] Amir S. Khan, Naseem A. Khan, and M.A. Syed, "A Software Tool for Learning the Inverse Kinematics of Robot Manipulators," *Journal of the Institution of Electrical and Electronics Engineers Pakistan, Vol. XXXIII,* July-December 1995.

[6] John J. Craig, **Introduction to Robotics: Mechanics and Control**, Second Edition, Addison-Wesley Publishing Company, 1989.

[7] Shimon Y. Nof, **Handbook of Industrial Robotics,** John Wiley & Sons, 1985.

[8] K. S. Fu, R. C. Gonzalez, and C. S. G. Lee, **Robotics: Control, Sensing, Vision, and Intelligence**, McGraw Hill Book Company, 1988.

[9] S. Harrington, **Computer Graphics: A programming Approach**, McGraw Hill Book Company, 1987.

[10] William M. Newman, **Principles of Computer Graphics**, McGraw Hill Book Company, 1983.

[11] Syed M. Ahmad, Abdul R. Khan, and Mahmood Ahmad, "Solving Kinematics of Serial Link Robot Manipulators Using Symbolic Computing," *Paper submitted for publication in Journal of the IEEEP.*