The Development and Implementation of a Robot Visualization System for Windows

Zhuang H.J. [1] [2], **Bauer F.** [1], **Khan W.A.** [1], **Angeles J.** [1]

[1]Department of Mechanical Engineering & Centre for Intelligent Machines
McGill University, Montreal, Canada
[2]Research Center of CAD/CAM Engineering
Nanjing University of Aeronautics and Astronautics, Nanjing, China

1 Introduction

Simulation and animation play an important role in the design stage of either a new robot or the whole work environment of an existing robot. RVS4W, the Robot Visualization System for Windows, is a software package based on RVS, that was developed at McGill University's Centre for Intelligent Machines in the 1990s [1]. Both RVS4W and RVS were developed for educational and research use. They are not commercial. RVS was developed on IRIX, the UNIX dialect of SGI, which is not compatible with most current operating systems. Therefore, the source code of RVS had to be modified in order to use this tool on an Intel-based PC. Based on the two important attributes of RVS4W, i.e., open source and platform-independent, the proper development tools were chosen. In this project, Dev-C++, OpenGL and Fast and Light Toolkit (FLTK) were adopted as the integrated development environment, graphics library and graphical user toolkit, respectively [2]. OpenGL builds the geometric object from simple primitives. It is possible to build a fully rendered robot by defining primitive after primitive and create all the required surfaces. But OpenGL is not efficient to create robotic architectures. Therefore, intermediate-level primitives e.g. cylinder , pivot and wheel are provided by RVS4W. The final robot design is built using a combination of these primitives.

2 Robot Animation

Robot animation is an indispensable function in robot visualization software. A kinematics engine is required to solve the basic displacement problems, i.e., the inverse and direct displacement problems. One robust algorithm for these problems was already included in the original RVS.
Calculating a new link position in Cartesian space is only half the task in robot animation. The remaining half is to display the animation in the OpenGL scene. This seems to be a simple task because it is one do-loop to set the new postures and redraw the model. However, this is impossible because of the FLTK routine used to redraw the OpenGL scene. In one do-loop of FLTK, only the first and last postures are rendered. Therefore, the simulation would show the robot "jump" from the initial to the final posture. The solution to this problem is the idle callback. This function is called every time the FLTK program waits for a new event. For RVS4W, the callback is initialized when the OpenGL window is created. An if-statement for each type of animation is defined inside the callback and the animation will be executed when the condition is true.

3 Optimum Design of Robots Using Gradient Methods

The design of a new robot starts with the determination of performance specifications that should be achieved by the robot. On the basis of these specifications, the various links are then designed. The minimization of the condition number of the Jacobian matrix over the feasible robot postures is supported by RVS4W.

3.1 Mathematical Background

We assume that the reader is familiar with the concepts of Denavit-Hartenberg parameters, condition number and Jacobian Matrix. Moreover, the concept of homogenous space, introduced in [2], is at the core of the design approach. In robotics the Jacobian matrix $\mathbf{J}$ is defined as

$$\mathbf{J}\dot{\theta} = \mathbf{t}$$

where

$$\mathrm{J} = \begin{bmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \cdots & \mathbf{e}_n \\ \mathbf{e}_1 \times \mathbf{r}_1 & \mathbf{e}_2 \times \mathbf{r}_2 & \cdots & \mathbf{e}_n \times \mathbf{r}_n \end{bmatrix}$$

in which $\mathbf{e}_i$ is the unit vector in the direction of the axis of the ith revolute joint and $\mathbf{r}_i$ is the vector, directed from the origin O_i of the ith DH frame or from any point of the foregoing axis for the matter, to the operation point P of the end-effector.
The condition number of a matrix $\mathbf{A}$ is defined based on the matrix Frobenius norm, namely,

$$\kappa_F(\mathbf{A}) = \frac{1}{n}\sqrt{\mathrm{tr}(\mathbf{A}\mathbf{A}^T)\mathrm{tr}[(\mathbf{A}\mathbf{A}^T)^{-1}]}$$

The condition number cannot be calculated for robots intended for both positioning and orienting tasks because the Jacobian matrix contains entries with nonhomogeneous dimensions. To solve this problem, vectors and distances are defined in homogenous space [2].

3.2 Optimization Problem

The approach assumed at the outset is the minimization of the robot condition number over its architecture parameters and joint variables. For a n-axis robot, $4n$ design parameters are available, over which the designer can minimize the condition number of the Jacobian matrix. Three of these parameters do not influence the condition number. The remaining $4n-3$ design variables are grouped in a design vector $\mathbf{x}$ in the form[1]

$$\mathbf{x}=[\ \bar{a}_1 \quad \alpha_1 \quad \bar{a}_1 \quad \bar{b}_1 \quad \alpha_2 \quad \theta_2 \quad \cdots \quad \bar{a}_n \quad \bar{b}_n \quad \theta_n\ ]^T$$

The optimum design problem is then defined as

$$\min_x \kappa_F(\mathbf{H})$$

subiect to the constraints

$$||\mathbf{e}_i|| = 1, \qquad i = 2, ..., n \tag{1}$$

$$\mathbf{e}_i \cdot \bar{\mathbf{r}}_i = 0, \qquad i = 1, 2, ..., n \tag{2}$$

in which $\mathbf{H}$ is the homogeneous, i.e., dimensionless, Jacobian matrix.

3.3 Solution Method

The problem is defined as [3]:

$$f(\mathbf{x}) \rightarrow \min_{\mathbf{x}}$$

Subject to the constraints

$$\mathbf{h}(\mathbf{x}) = 0$$

where

$$f(\mathbf{x}) = \frac{1}{n}\sqrt{\mathrm{tr}(\mathbf{H}\mathbf{H}^T)\mathrm{tr}[(\mathbf{H}\mathbf{H}^T)^{-1}]}$$

Vector $\mathbf{h}$ includes the left-hand sides of constraints (1) and (2), namely,

$$\mathbf{h}=\begin{bmatrix} \|\mathbf{e}_2\|^2 - 1 \\ \vdots \\ \|\mathbf{e}_n\|^2 - 1 \\ \mathbf{e}_1 \cdot \bar{\mathbf{r}}_1 \\ \vdots \\ \mathbf{e}_n \cdot \bar{\mathbf{r}}_n \end{bmatrix}$$

The objective function can be simplified as

$$z = \mathrm{tr}(\mathbf{H}^T\mathbf{H})\mathrm{tr}[(\mathbf{H}^T\mathbf{H})^{-1}] \rightarrow \min_{\mathbf{x}}$$

which is factored into two functions, namely,

$$f_1 = \mathrm{tr}(\mathbf{H}^T\mathbf{H})$$
$$f_2 = \mathrm{tr}[(\mathbf{H}^T\mathbf{H})^{-1}]$$

This leads to the gradient

$$\nabla z = f_2\nabla f_1 + f_1\nabla f_2$$

The process of computing gradient of the objective function f_1 and f_2 is omitted due to space limitation.

4 Conclusions

RVS4W offers the same functionality on an Intel PC as the original RVS on IRIX. Some new functions are included in RVS4W.

References

[1] Wu, C.J., Angeles, J., and Montagnier, P. *Robot Visualization System (RVS) User Manual*, Centre for Intelligent Machines (CIM) Department of Mechanical Engineering McGill University, Canada, 1997.

[2] Angeles J., *Fundamentals of Robotic Mechanical Systems: Theory, Methods, and Algorithms*, Third Edition, New York, 2007.

[3] Bauer F. *The Development and Implementation of Kinematics Algorithms on RVS*, Master Thesis, Fachhochschule Münster, Münster, Germany, 2006.

[1] $\bar{a}_i$ and $\bar{b}_i$ denote DH parameters in homogenous space; They are dimensionless.