Submitted to MathMod Vienna, Feb. 2-4, 1994, Vienna, Austria.

Integrated Modeling, Simulation, and Animation of Rigid Arms and Vehicles through Object-Oriented Programming

Paul R. Schmitt John E. Hogan Jonathan M. Cameron Wayne J. Book

Georgia Institute of Technology
The George W. Woodruff School
of Mechanical Engineering
Atlanta, GA 30332
USA

Abstract. This paper describes an object-oriented system, "MBSIM" (MultiBody SIMulator), that models, simulates, and animates the kinematics and dynamics of robotic arms and vehicles. This system creates a three-dimensional graphical environment which can be used as a tool in robotic design and control.

1. INTRODUCTION

A motivation for developing MBSIM arises from an inspection task in the nuclear industry. This task requires vehicles to move safely along narrow aisles to inspect drums of low-level radioactive waste. Safe motion requires sensor-based motion control to avoid collisions with walls and obstacles. Given the difficult nature of the environment involved, a simulation system is necessary for the development and testing of motion planning and control algorithms.

We desired MBSIM to be a convenient and flexible tool for studying the motions of various mechanisms including vehicles, robotic arms, and combinations of the two. Our simulation integrates the kinematics, dynamics, sensors, and graphics of mechanisms into a modular environment. Each mechanism consists of a series of links. The mechanism's kinematics and dynamics are determined from the relative kinematics and mass properties encapsulated in each link. Sensors can be attached onto each link object as desired. These sensors are useful for path generation and control routines. Mechanism graphics are constructed from graphics descriptions associated with each link object.

The types of systems that MBSIM can model involve physical objects such as robots, vehicles, and obstacles, objects that share many characteristics. An object-oriented approach is appropriate for modeling these systems. Common characteristics such as position and geometric shape can be developed once and reused appropriately via object inheritance and inclusion. The resulting software objects reflect the modularity of the world and can be dealt with intuitively. In fact, one of the important benefits of the object-oriented approach is the ability to think of software objects in the same way as we do real objects. MBSIM utilizes object-oriented inheritance to construct new types of links from existing links. To illustrate some benefits of the object-oriented approach, this paper includes a case study that outlines the steps taken as well as the time involved in modeling and simulating a mobile platform.

2. SOFTWARE DESIGN

Our system is implemented using the object-oriented C++ language [4]. C++ is an object-oriented programming language that directly supports inclusion (using user-created objects as data) and inheritance

(designing objects which inherit previously developed data structures and code from more basic object definitions).

In MBSIM, each mechanism is constructed as a series of links. Each link incorporates the joint kinematics that connect it to the preceding link. In other words, each link "knows" how to move itself with respect to the previous link. This general joint paradigm allows holonomic and non-holonomic constraints to be treated similarly. This paradigm also facilitates modeling and simulation of the kinematics and dynamics of multibody mechanisms with a wide variety of link types. The links contain graphic objects which describe the physical shape of the link and sensors which provide feedback of the environment.

3. KINEMATICS

Following the joint paradigm, each link contains functions to compute its angular and linear velocity and acceleration relative to the preceding link. These functions are pure virtual functions in the generic link class and therefore must be defined in every specific link sub-class. This ensures a uniform interface to each specific link that can be used to calculate velocities and accelerations of any link. The velocities or accelerations are calculated by propagating the velocity out from the base link to the link of interest. This propagation from link to link requires the uniform interface so that the velocities or accelerations can be calculated for mechanisms composed from any number and types of links.

Each specific link also contains functions which enable velocity or acceleration joint variables to be integrated to yield the mechanism motion. These velocities or accelerations would be commanded from control schemes such as obstacle avoidance and path planning. These control schemes would generally be implemented in the simulation to make use of sensor feedback and dynamic characteristics of the system. The commanded velocity or acceleration may be output to an external file for further analysis or for future replays. If needed, this also enables data from external control schemes to be read and tested with the simulation. These integration functions in each link, one for velocities and one for accelerations, contain equations which relate joint velocity variables to joint position variables and joint acceleration variables to joint velocity variables, respectively. Inside these functions, constraint or auxiliary equations may be used to determine the specific behavior of the link, as in a non-holonomic link. The mechanism calls the integration function of each link in the same manner, whether the link is holonomic or non-holonomic.

4. INVERSE DYNAMICS

The inverse dynamics of the system is calculated using an iterative Newton-Euler dynamic formulation similar to the one found in [2]. The inverse dynamics yields the forces and torques that each link's actuator would need to generate motion based on each link's positions, velocities, and accelerations. This is useful to determine whether the desired velocity or acceleration commands are feasible for a particular actuator. Many control schemes such as those found on a Denning robot produce velocity or acceleration commands rather than forces or torques. The consistent velocity and acceleration interface contained within each link enables only one function at the mechanism level to calculate the dynamics for each link in any mechanism in any configuration. Once again, by embedding the specific characteristics of a link inside the specific link class and using consistent interfaces to each link, general functions can be used to yield desired quantities regardless of the mechanism being studied.

5. SENSOR MODELING

To model a mechanism in an unknown or partially known environment, we require sensor feedback of the surroundings. Our system currently incorporates an idealized rangefinder model. As previously mentioned the link sensor file holds the positions and rotations of each sensor on the link. This file is flexible, allowing the user to place an arbitrary number of sensors in any position or orientation on any link.

When activated, the sensor currently scans through all objects in the environment to determine whether the objects' enclosing spheres lie on the sensor's line of sight. Once this rough reading has been determined, the

object's actual geometric characteristics are tested for intersection with the sensor's line of sight. This two-stage method reduces computational overhead.

6. IMPLEMENTATION

The MBSIM hierarchy, as developed above, is implemented into four main parent classes: Mechanism, Link, Sensor, and Graphics_Object. The partial class structure in Fig. 1 shows their inheritance and inclusion relationships. These classes are constructed from script files external to the program. These files are read at run-time so additions and modifications of a wide variety of mechanisms can be made without having to re-compile MBSIM.

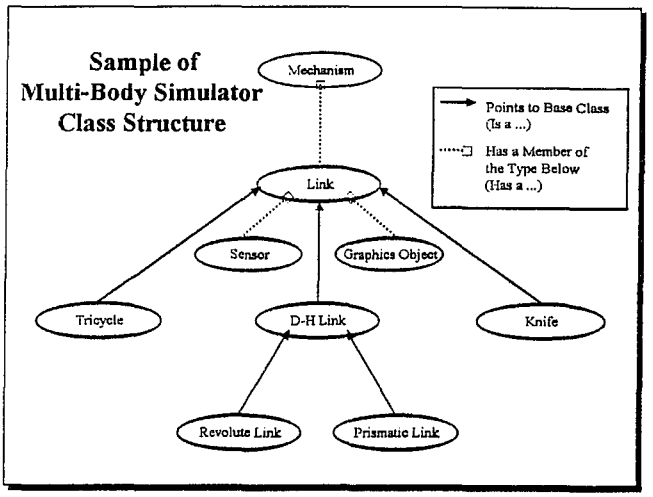

Fig. 1. MBSIM class structure sample.

The Graphics_Object class encapsulates all graphics library function calls. The system initially utilized SPHIGS, a public domain software library for 3-dimensional graphical display [3]. (SPHIGS is a simplified implementation of the well known 3-dimensional graphics protocol, PHIGS.) This encapsulation facilitates improvements to the graphics library.

7. CASE STUDY

A nuclear industry testbed vehicle is driven by two parallel wheels on a fixed axis with independent velocity control. The vehicle, as shown in Fig. 2 is equipped with twenty-four ultrasonic sensors.

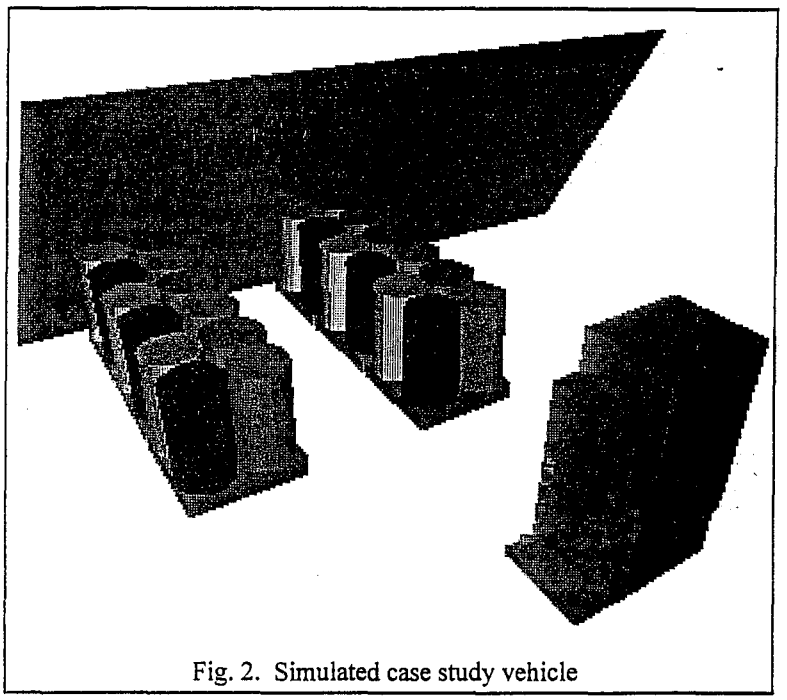

Fig. 2. Simulated case study vehicle

The following steps were taken to simulate this vehicle. The manual derivation of the vehicle's kinematic and dynamic link equations required one hour. These equations were incorporated into a new class derived from the parent Link class, requiring a second hour. The vehicle took shape as its dimensions were set into a graphics description file requiring a half hour. Next, sensors were added to the vehicle through a sensor description file. Positioning and orienting the twenty-four sensors took an hour. Another hour was needed to develop a routine to test the accuracy of the above files and code. The total time required to develop this new link and tailor it to a specific vehicle was about four and one half hours.

8. CURRENT WORK

Current work is focused on extending the flexibility of MBSIM. These areas include forward dynamics, sensor modeling, sensor data mapping, and graphics library improvements.

Forward dynamics can be determined by putting special values into the iterative Newton-Euler routine to determine the mass matrix and the non-linear Coriolis, centrifugal, and gravity terms. Once found, one can then

use these matrices to solve for joint accelerations given joint torques. Velocity and position are then obtained by integrating acceleration.

An ultrasonic sensor is being modeled. Where appropriate, noise will be added to the returned data to more accurately simulate real sensor data.

An intelligent sensor data storage array map is being developed to handle sensor input. Histogramic in-motion mapping will be used mainly to record levels of obstacle existence evidence [1]. The map updates data from new readings and will be used to create and evaluate obstacle avoidance paths.

Currently, the graphics routines limit the performance of our system. This is due to our hardware and the simple nature of the SPHIGS graphics library. While excellent for our initial learning phase, SPHIGS does not have the performance of commercial graphics libraries. We are upgrading our hardware and implementing a more advanced and efficient graphics library.

9. CONCLUSION

MBSIM, an object-oriented three-dimensional robotic simulator, has been introduced. This simulation integrates the kinematics, dynamics, sensors, and graphics of mechanisms into a modular environment. A case study that outlines the steps taken and time involved in modeling and simulating a particular mobile was presented to demonstrate the system's flexibility. Current work will enhance the system's performance, flexibility, and sensing the environment.

10. ACKNOWLEDGMENTS

This work was partially supported by subcontract 92006 to the Educational Research and Development Association (ERDA) of Georgia Universities under contract with Westinghouse Savannah River Plant.

11. REFERENCES

[1] Borenstein, Johann and Koren, Yoram, Histogramic In-Motion Mapping for Mobile Robot Obstacle Avoidance, *IEEE Transactions on Robotics and Automation*, vol. 7, no. 4, August 1991, pp. 535-539.

[2] Craig, John, Introduction to Robotics: Mechanics and Control, 2nd Ed., Addison-Wesley Publishing, 1989.

[3] Foley, van Dam, Feiner, and Hughes, Computer Graphics - Principles and Practice, Addison-Wesley Publishing, 1990.

[4] Stroustrup, Bjarne, The C++ Programming Language, 2nd Edition, Addison-Wesley Publishing, 1991.