Computer Simulation of Interacting Dynamic Mechanical Systems using Distributed Memory Parallel Processors

Michael C. Stanley
James E. Colgate
Department of Mechanical Engineering
Northwestern University
Evanston, IL 60208

Abstract

A high performance parallel processing architecture has been designed for the simulation of dynamic mechanical systems. This architecture is designed to exploit parallelism in the simulation of rigid body motion in order to achieve real-time performance. The it is also capable of detecting and simulating collisions between bodies in the system. A real-time control system is incorporated for providing realistic force feedback to a human subject using a four degree of freedom manipulandum. The computer architecture also provides an interface for real-time graphics display of the mechanical simulation. The design is based on the use of the Inmos transputer, a compact microprocessor specially designed for parallel processing architectures.

Introduction

The development of compact, multitasking microprocessors has made the implementation of distributed memory parallel processing architectures possible[16, 21]. The computational power provided by these architectures allows the real-time solution of computationally intensive systems of equations. The real-time solution of the dynamic equations of motion for complex mechanical systems allows us to implement a simulation system with a human operator in the simulation loop. This paper describes the design of a system capable of providing a human subject with realistic force feedback from the real-time simulation of dynamic mechanical systems. The system is designed to simulate a wide variety of two-dimensional environments, including those involving collisions between objects. It also provides real-time control for a four degree of freedom manipulandum[15], and displays a graphical representation of the mechanical system. Potential uses include creation of virtual realities, teleoperation, and macro-micro manipulation.

Section I describes the attributes of the Inmos transputer that make it useful for parallel system implementations. Section II discusses difference in software design philosophy for parallel processing architectures. Section III describes the method used to simulate dynamic systems. Section IV describes the collision detection and response issues. Section V describes the design of the simulation and control architecture.

I. Inmos Transputers

The basic component of the system is the Inmos T805 Transputer[2]. The T805 is a 32-bit microprocessor specifically designed for parallel processing applications. The processor runs at a 30 MHz clock speed and provides a sustained instruction rate of 15 MIPS (30 MIPS peak). The 64-bit floating point coprocessor provides a sustained rate of 3.3 MFLOPS (4.3 MFLOPS peak).

The T805 processor was specifically designed to support the execution of concurrent or parallel processes. The processor incorporates an efficient process scheduler that allocates processor time and maintains process information. Processes may be run at high or low priority and inactive tasks do not consume any processing time. The low-level implementation of the scheduler provides process switching times of less than 1 microsecond.

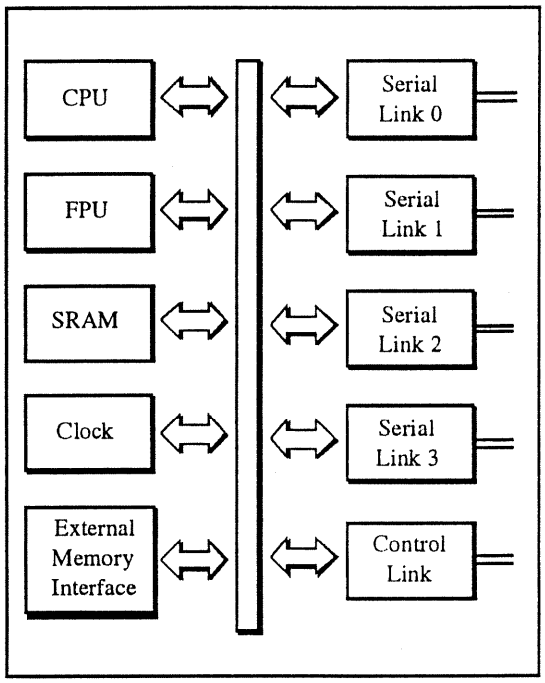

Figure 1: The Inmos Transputer Chip

Interprocessor communication is achieved using 4 high speed bi-directional serial links. Each link pair has an on chip DMA interface and communication support for the links is built into the microprocessor instruction set. The links operate at a rate of 20 Mbits/sec and can achieve a throughput of 2.4Mbytes/sec. Connections are established by directly connecting serial links between transputers. Alternatively, Inmos supplies a 32x32 solid state switch, called a configuration link switch, that connects transputer links under software control.

Serial links are also used for booting the transputer network. When power is initially applied to the network, each transputer looks to obtain boot information from any of the serial links. Networks are booted by loading an initialization program onto one processor that copies itself throughout the network.

Additional features of the T805 include 4 kbytes of on chip SRAM with a 50-ns cycle time and an on chip interface for accessing up to 4 Gbytes of external DRAM. The Inmos transputer chip is usually bundled with supporting circuitry such as external DRAM, serial to parallel converter, etc. These bundled units are called transputer modules or TRAMs[3]. Inmos also supplies "motherboards" which are printed circuit boards that supply power and service signals to the TRAMs and mount in several standard computers.

II. Parallel Programming

The low-level support for concurrent processing using transputer architectures leads to changes in the design philosophy when writing software for parallel processing networks. This change in philosophy is embodied in OCCAM[1], a programming language specifically designed to support the use of concurrent processes. Although transputers can be programmed in other high-level languages such as C, OCCAM provides low level constructs for creating parallel processes and setting up communication between them.

Software design for concurrent execution is based on the notion of the process. Each process is an independent unit consisting of a sequence of tasks. Processes may be built up from other processes, so that system design is hierarchical. Processes communicate using communication links called channels. A channel identifies the communicating processes and the type of data it transfers. Channels communicate only when both processes are ready. Thus, channel communication is synchronous and a separate method for synchronizing processes is not needed.

Channels may be used independent of the actual physical location of the communicating processes. If two processes are executing on the same transputer, then channel communication is achieved using memory to memory data transfer. If two processes are executing on different transputers, then channel communication is achieved using one of the transputer's serial links (see Figure 2). External and internal channels are treated identically during code development.

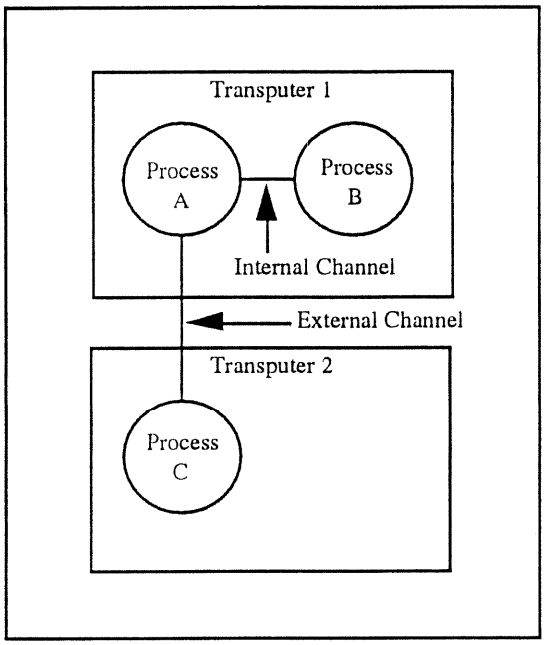

Figure 2: Channel Communication

Because the actual location of the other process communicating over the channel is transparent to the communicating processes, software can be developed independent of the actual hardware configuration. Knowledge of the specific hardware configuration is needed only at runtime when the processes are allocated to specific processors and the physical means of communication is specified for each channel. Because process communication is independent of the actual hardware configuration, systems can be easily reconfigured to meet performance requirements.

III. Dynamics Simulation

Various methods for formulating the dynamic equations of motion for mechanical systems have been presented in literature[5, 6, 8, 13, 14, 19, 22, 23, 24]. Most are based on the creation of a library of joints between bodies in the system. A simulation environment is created by specifying objects and object properties and then connecting the objects with joints from the library. The objects and their connections are used to formulate a system of equations which is then solved for the accelerations of the bodies. These accelerations are numerically integrated to obtain object velocity and position at the next time step.

Since we are interested in generating realistic force feedback to a human subject, it is essential for the system to be able to simulate a wide range of dynamic behaviors. The most difficult behavior to simulate is a stiff system. Previous experience with a one degree of freedom manipulator suggest that force update rates of several hundred hertz are necessary to achieve reasonable stiffness.

We plan to use a simplistic formulation of the system dynamics where each connection consists of a generalized spring and damper. Examples of these generalized springs and dampers (referred to as "actions") are given in the Appendix. Although more rigorous formulations are possible[5, 6, 8, 13, 14, 19, 22, 23, 24], using springs and dampers to simulate connections decouples the system dynamic equations of motion. The acceleration of each object can then be computed independently. Thus, using springs and dampers leads to a very low level of parallelism in solving the equations of motion. Using springs and dampers also lets us handle collisions in a simple manner. Collisions are simulated by adding a spring and damper at the point of collision and then proceeding with the simulation.

There are several disadvantages to this approach. In order to create a rigid connection between objects, very stiff springs must be used. The resulting equations must use a small integration time step in order to maintain numerical stability. Since the equations of motion must be solved during each time step, the small step size imposed by the integration routine requires high computational performance. However, since the equations of motion are decoupled, we can easily distribute tasks over the parallel architecture and thus achieve short execution times.

The choice of stiffness and damping parameters for a given connection is also difficult. A possible solution is to continuously adapt these parameters to maintain numerical stability of the integration routine. This may be done by monitoring the integration error at each time step and increasing or decreasing the joint stiffness based on the change in the error.

Simulation of the environment is achieved by assigning a transputer to each object in the environment. Since we are using generalized springs and dampers to simulate connections between objects, all external forces acting on the object depend only on the states of the object and the objects connected to it. Thus, by configuring the transputer network to mirror the topology of the actual physical system, the states of connected objects can be passed over the transputer's serial links. Then, each transputer can calculate the external forces acting on the object it represents. Once the external forces are determined, the object accelerations may be calculated and integrated to determine new object position and velocity.

The number of connections to other objects is limited only by the number of serial links available to the transputer. The T805 has four serial links, one of which is used to transfer object information to collision detection and graphics. Thus, each object may have up to three other objects connected to it. Mechanisms with kinematic loops may be treated identically to serial mechanisms. Any loops are automatically incorporated into the simulation when the transputer links are configured.

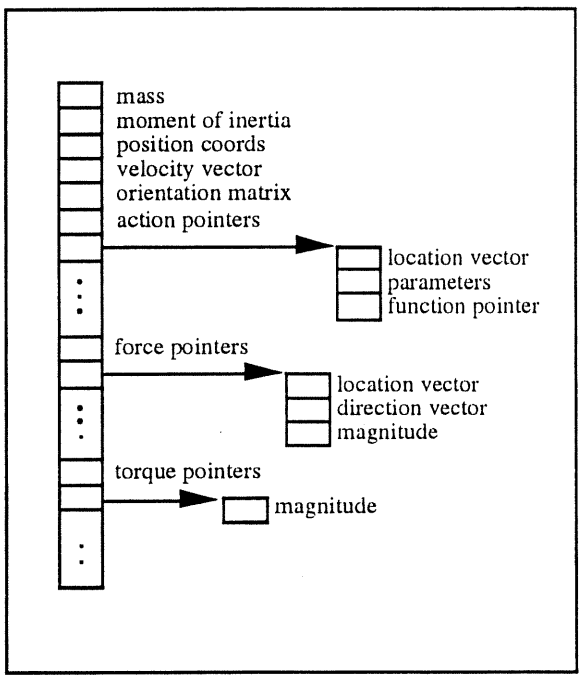

Figure 3: Object Data Structure

Data organization parallels the structure described in [24]. Object information is stored in the data structure shown in Figure 3. The object data structure contains information about the object (mass, moment of inertia, ...), the object states, and arrays of pointers to actions, forces, and torques that act on the object. Each action data structure contains a vector from the center of mass to the point of application, parameters necessary for its evaluation, and pointers to functions that evaluate its value (See Appendix A). Force and torque data structures are similar to the action structure. The object data structure provides a simple means for simulating the object. For each time step, the transputer indexes through the arrays of actions, forces, and torques, calculating and summing the forces and torques acting on the object. Once all forces are calculated, the object acceleration can be determined.

The object data structure also allows the addition of new actions due to collisions with other objects. This is accomplished by creating a new action data structure and adding another pointer to it to the object's action array.

IV. Collision Detection and Response

Simulation of interacting dynamic systems necessarily involves the detection of collisions between objects and response to these collisions. Algorithms employed in detecting collisions are specific to the method used to represent object boundaries. Object boundaries are commonly represented as polygons in two dimensions or as a set of polyhedra in three dimensions[10, 12, 18]. Other representations are possible but require a large amount of computation[7].

Algorithms for collision detection between arbitrary polygons in two dimensions are highly developed [4, 9, 11, 20]. These algorithms reduce the number of edges that must be checked for intersection by first determining those edges visible from a test point. The remaining visible edges are then ordered so that the search algorithm achieves better than linear performance in the number of vertices.

Collision response techniques have previously been used to generate realistic animation of colliding objects [7, 17]. These methods use a

hybrid method for determining system response to collisions. Object velocities after a collision are determined analytically from the velocities before the collision and a model of collision dynamics. Springs are also used for mild collisions, such as an object resting on another object, so that the analytical solution does not have to be continuously computed.

V. System Architecture

We are in the process of implementing a comprehensive system for the simulation of dynamic mechanical systems. This system consists of a high-performance manipulandum, graphic display device, and the computer simulation and control system. The system will be used to determine the requirements of a realistic force simulation system as well as for uses in teleoperation and micro-manipulation with force feedback.

The system may be organized into five logical subsystems: control, simulation, collision detection, graphics, and development.

Control Subsystem

The control subsystem generates torque commands to the manipulandum and determines manipulandum position from joint angles. Motor torques are calculated from endpoint torques and forces received from the simulation. The manipulandum position is used as input to the simulation and is also monitored to make sure the manipulandum is not moved to an invalid configuration.

Simulation Subsystem

The simulation subsystem uses manipulandum position to calculate new object positions, velocities, and accelerations as well as the corresponding joint forces and torques. Forces and torques are sent back to the control subsystem for output to the manipulandum. The new object positions are also output to the graphics and collision detection subsystems.

Collision Detection Subsystem

Polygons are used to represent objects in the collision detection subsystem. Object positions are received from the simulation subsystem and used to determine if a collision has occurred. If a collision is detected, information about the location of the collision and the objects involved is relayed back to the simulation subsystem so that it can be incorporated into the simulation.

Graphics Subsystem

The graphics subsystem displays a graphical representation of the system to the user. The system currently uses the same polygonal representation as the collision detection subsystem, however, other representations may also be used to create a more realistic interface. Since the graphics display subsystem is separate from collision detection, the complexity of the graphical display does not affect collision detection update rates.

Development Subsystem

The development subsystem provides capabilities for developing software and simulation environments, monitoring and controlling simulations, and storing and analyzing data. This subsystem is not involved in any real-time simulation activities.

VI. Conclusions

A system for the real-time simulation of dynamic mechanical systems has been described. This system will be used to investigate dynamic interaction between a human operator and a usefully complex virtual environment. Computational performance and psychophysical results will be reported in future publications.

VII. Acknowledgments

The authors gratefully acknowledge the support of the National Science Foundation, Grant #MSS-9022513.

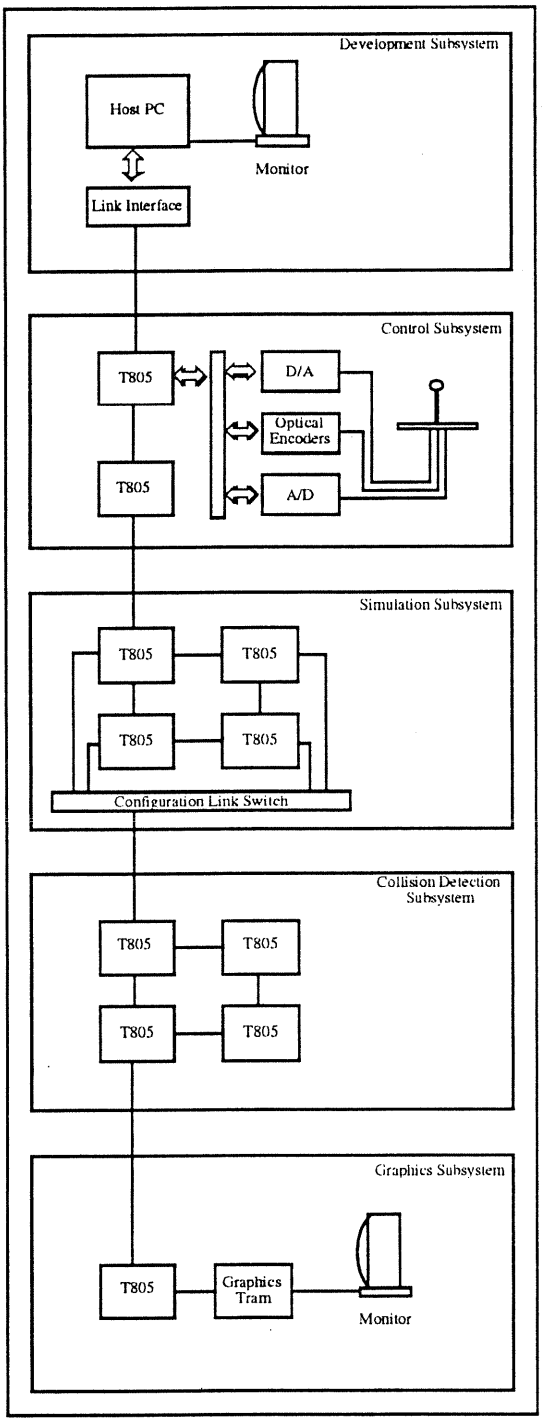

Figure 4: System Architecture

VII. References

1. "OCCAM 2 Reference Manual." Prentice Hall International Series in Computer Science. Hoare ed. 1988 Prentice Hall.

2. "The Transputer Databook." 1989 Inmos Ltd.

3. "The Transputer Development an iq Systems Databook." 1991 Inmos Ltd.

4. Avis, D. and G. T. Toussiant. An Optimal Algorithm for Determining the Visibility of a Polygon from an Edge. IEEE Transactions on Computers. **Vol. C-30**(No. 12): 1981.

5. Bae, D. and E. J. Haug. A Recursive Formulation for Constrained Mechanical System Dynamics: Part I. Open Loop Systems. Mechanical Structures and Machines. **Vol. 15**(No. 3): 359-382, 1987.

6. Bae, D., J. Kuhl and E. J. Haug. A Recursive Formulation for Constrained Mechanical System Dynamics: Part III. Parallel Processor Implementation. Mechanical Structures and Machines. **Vol. 16**(No. 2): 249-269, 1988.

7. Baraff, D. Curved Surfaces and Coherence for Non-penetrating Rigid Body Simulation. Computer Graphics. **Vol. 24**(No. 4): 1990.

8. Barzel, R. and A. Barr. A Modeling System Based on Dynamic Constraints. Computer Graphics. **Vol. 22**(No. 4): 1988.

9. Chin, F. and C. A. Wang. Optimal Algorithms for the Intersection and Minimum Distance Problems Between Planar Polygons. IEEE Transactions on Computers. **Vol. C-32**(No. 12): 1983.

10. Cyrus, M. and J. Beck. Generalized Two- and Three- Dimensional Clipping. Computers and Graphics. **Vol. 3**: pp. 23-28, 1978.

11. Dobkin, D. P. and D. G. Kirkpatrick. A Linear Algorithm for Determining the Seperation of Convex Polyhedra. J. Algorithms. **Vol. 6**: pp 381-392, 1985.

12. Griffiths, J. G. Bibliography of Hidden-line and Hidden-surface Algorithms. Computer Aided Design. **Vol. 10**(No. 3): pp. 203-206, 1978.

13. Haug, E. J. "Elements and Methods of Computational Dynamics." Computer Aided Analysis and Optimization of Mechanical System Dynamics. Haug ed. 1984 Springer-Verlag.

14. Luh, J., M. Walker and R. Paul. On-Line Computational Scheme for Mechanical Manipulators. Journal of Dynamic Systems, Measurement, and Control. **Vol. 102**: 1980.

15. Millman, P. and J. Colgate. Design of a Four Degree-of-Freedom Force-Reflecting Manipulandum with a Specified Force/Torque Workspace. Proceedings of the 1991 IEEE Conference on Robotics and Automation. **Vol. 2**: 1991.

16. Mindell, D. and D. Yoerger. Transputer-Based Distributed Processing for Underwater Robotic Vehicle Control. Proceedings of the 1991 American Control Conference. **Vol. 1**: 1991.

17. Moore, M. and J. Wilhelms. Collision Detection and Response for Computer Animation. Computer Graphics. **Vol. 22**(No. 4): 1988.

18. Rogers, D. "Procedural Elements for Computer Graphics." 1985 McGraw-Hill Book Company. New York.

19. Schroder, P. and D. Zeltzer. The Virtual Erector Set: Dynamic Simulation with Linear Recursive Constraint Propagation. Computer Graphics. **Vol. 24**(No. 2): 1990.

20. Schwartz, J. T. Finding the Minimum Distance Between Two Convex Polygons. Information Processing Letters. **Vol. 13**(No. 4-5): 1981.

21. Shoemaker, R., H. Barrett, A. Landesman, R. Seacat and B. Taylor. The TRIMM Parallel Processor. Computers in Physics. : 1991.

22. Walker, M. and D. Orin. Efficient Dynamic Computer Simulation of Robotic Mechanisms. Journal of Dynamic Systems, Measurement, and Control. **Vol. 104**: 1982.

23. Wang, L. and B. Ravani. Recursive Computations of Kinematic and Dynamic Equations for Mechanical Manipulators. IEEE Journal of Robotics and Automation. **Vol. RA-1**(No. 3): 1985.

24. Witkin, A., M. Gleicher and W. Welch. Interactive Dynamics. Computer Graphics. **Vol. 24**(No. 2): 1990.

Appendix

The following derivations and notation are taken from [13].

Planar Dynamic Equations of Motion

The differential equations of motion for a rigid body i in a plane can be written as

$$m_i \ddot{r}_i = F_i \qquad (1)$$

$$J_i \ddot{\phi}_i = \tau_i \qquad (2)$$

where m_i is the mass of body i, $r_i = [x_i, y_i]^T$ is the position of the centroid of body i, F_i is a vector of external forces acting at the centroid, J_i is the moment of inertia of body i, ϕ_i is the angular orientation, and τ_i is the applied external torque.

This may be written as

$$M_i \ddot{q}_i = Q_i \qquad (3)$$

where $M_i = \text{diag}(m_i, m_i, J_i)$, $q_i = [x_i, y_i, \phi_i]^T$, and Q_i is a vector of generalized forces associated with the external forces and torques acting on body i.

The form of Q_i is specific to the type of applied force or torque. Examples of Q_i for some common forces and torques are given in the next section.

Fixed Force

Consider the force F acting at point P on body i as shown in Figure 1. The generalized force vector due to this force may be written as

$$Q_i = \begin{bmatrix} F \\ s_p^T B_i^T F \end{bmatrix} \qquad (4)$$

where s_p is a vector from the point of application of the force to the centroid described in a body fixed reference frame, and

$$B_i = A_i R = \begin{bmatrix} \cos\phi_i & -\sin\phi_i \\ \sin\phi_i & \cos\phi_i \end{bmatrix} \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \qquad (5)$$

A_i is the transformation matrix from the global reference frame to the body fixed reference frame and R an orthogonal rotation matrix.

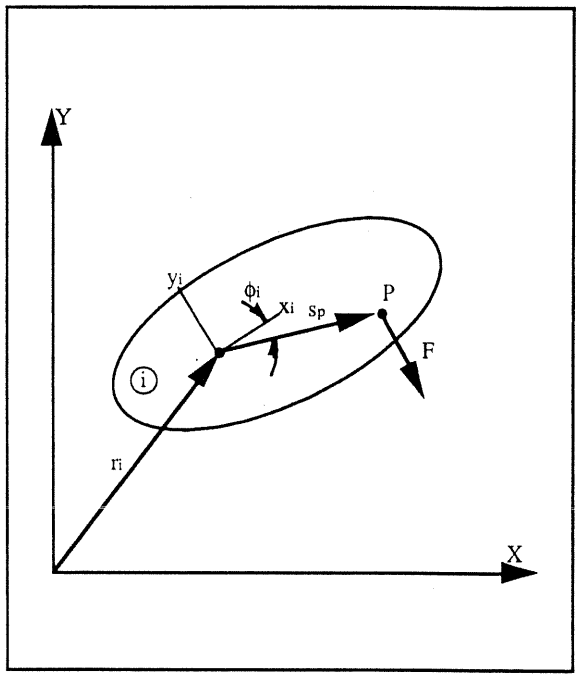

Figure 1: Fixed force acting on body i.

Translation Spring-Damper-Actuator

Consider the translational spring-damper-actuator acting between points P_i and P_j on bodies i and j as shown in Figure 2. The generalized force vector due to this element may be written as

$$Q_i = \frac{f}{l} \begin{bmatrix} d_{ij} \\ d_{ij}^T B_i s_i \end{bmatrix} \qquad (6)$$

$$Q_j = -\frac{f}{l} \begin{bmatrix} d_{ij} \\ d_{ij}^T B_j s_j \end{bmatrix} \qquad (7)$$

where

$$\mathbf{d}_{ij} = \mathbf{r}_j + \mathbf{A}_j\mathbf{s}_j - \mathbf{r}_i - \mathbf{A}_i\mathbf{s}_i \qquad (8)$$

is a vector from point P_i on body i to point P_j on body j,

$$l = \sqrt{\mathbf{d}_{ij}^T\mathbf{d}_{ij}} \qquad (9)$$

is the distance between points P_i and P_j,

$$\dot{l} = \left(\frac{\mathbf{d}_{ij}}{l}\right)^T\left(\dot{\mathbf{r}}_j + \mathbf{B}_j\mathbf{s}_i\dot{\phi}_j - \dot{\mathbf{r}}_i - \mathbf{B}_i\mathbf{s}_i\dot{\phi}_i\right) \qquad (10)$$

is the rate of change of this distance, and

$$f = k(l - l_0) + b\dot{l} + F(l, \dot{l}, t) \qquad (11)$$

is a general force function where k is the spring coefficient, l_0 is the free length of the spring, b is the damping coefficient, and F is a general actuator force which may be a function of l, $\dot{l}$, and time t.

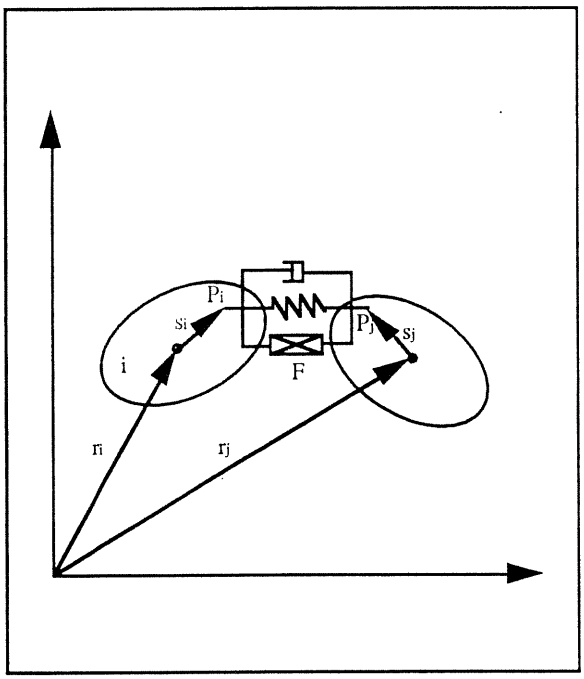

Figure 2: Translational spring-damper-actuator

Rotational Spring-Damper-Actuator

Consider the rotational spring-damper-actuator acting between body fixed axes xi and xj as shown in Figure 3.

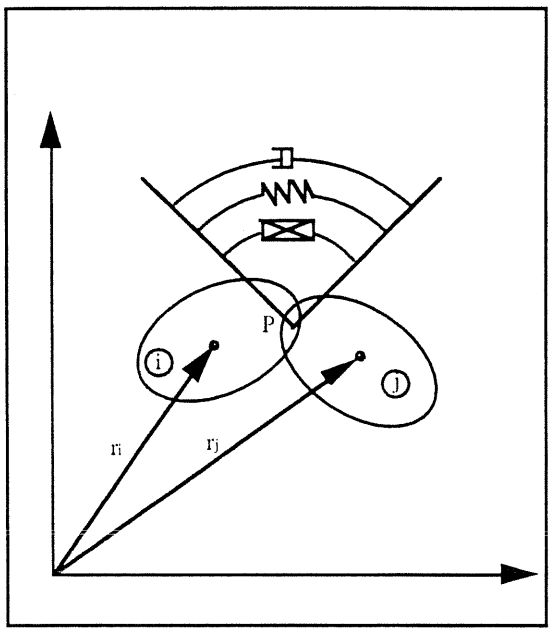

Figure 3: Rotational spring-damper-actuator

The generalized force vector due to this element may be written as

$$Q_i = \begin{bmatrix} 0 \\ 0 \\ \tau \end{bmatrix} \qquad (12)$$

$$Q_j = -\begin{bmatrix} 0 \\ 0 \\ \tau \end{bmatrix} \qquad (13)$$

where

$$\tau = k(\theta_{ij} - \theta_0) + b\dot{\theta}_{ij} + N(\theta_{ij}, \dot{\theta}_{ij}, t) \qquad (14)$$

is a general forcing function, $\theta_{ij} = \phi_j - \phi_i$ is the rotational difference between the axes, $\dot{\theta}_{ij} = \dot{\phi}_j - \dot{\phi}_i$ is the rate of change of the difference, k is the

spring coefficient, θ_0 is the free angle of the spring, b is the damping coefficient, and N is a general actuator force which may be a function of θ, $\dot{\theta}$, and time t.