Blender for robotics and robotics for Blender

Herman Bruyninckx,*
Dept. of Mechanical Engineering
K.U.Leuven, Belgium
http://www.orocos.org

Abstract

Computer animation and robotics have most of their mathematical foundations in common. So, Blender is a natural (but still undiscovered and imperfect) GUI candidate for robot simulation and programming. The robotics research community lacks an advanced graphical tool such as Blender, but, on the other hand, it has more advanced and efficient algorithms than Blender for the physically realistic simulation of armatures. For example, the inverse kinematis of humanoid structures, taking into account their dynamics, and with a generic approach for the automatic generation of natural motions. In this paper, I make concrete suggestions about additions to the Blender code base, in order to support these complex armatures and IPO's. My suggestions touch the following parts of Blender: *moto* (Motion Toolkit), *gameengine*, and (especially) *iksolver*.

1 Introduction

Robotics and computer animation have much of their mathematical background in common: 3D motion specification and visualisation; forward and inverse kinematics of "armatures"; interaction with the "outside world" via sensors and actuators; there exist some commercial programming and visualisation tools, but they are expensive and have a high degree of *lock-in*; there exist some other open source programs, but they miss one or more of the core components.

Blender seems to be, in the long term, a perfect match for all robotics needs. The reasons why it *will be* a perfect match are:

- It has a very good basic design, with reasonably well decoupled and accessible code.
- It contains most of the basic data structures and libraries for robotics. In casu, *moto* and *iksolver*.
- It contains good support for rendering of the simulated motion, including the generation of videos.
- It is *fast*.
- It is *customizable*.
- The "competition" in the robotics domain is very immature.

The reasons why it is *not yet* a perfect match are:

- Real physical units must be used everywhere, i.e., a *Blender unit* should be given a unique meaning, for example: 1 Blender unit = 1mm. Robotics is not just interested in something that *looks* realistically, it *has to be* realistic.
- The inverse kinematics solver is too simplistic: only a general-purpose algorithm is provided, and all joints are by default *spherical*. Robotics has multiple *armature families* for which efficient but dedicated algorithms exist; in addition, the one-dimensional *revolute* and *prismatic* joints are *the* building blocks in robotics (Fig. 1), but they are not basic building blocks in Blender's armatures.
- Robotics has multiple *IPO families* for *automatic* trajectory generation, i.e., not just spline interpolation between *key frames*.
- The Blender interface is *too difficult* for roboticists.

This paper gives a roadmap for the evolution of Blender towards a *must-have* tool for every robotics

*The author gratefully acknowledges the financial support by K.U.Leuven's Concerted Research Action GOA/99/04, and the OCEAN project of the European Union's IST programme.

research lab. Fortunately, most of this roadmap contains most probably also developments that the non-robotics Blender community would like to see happen. The core of this roadmap is about easier and more natural generation of motions.

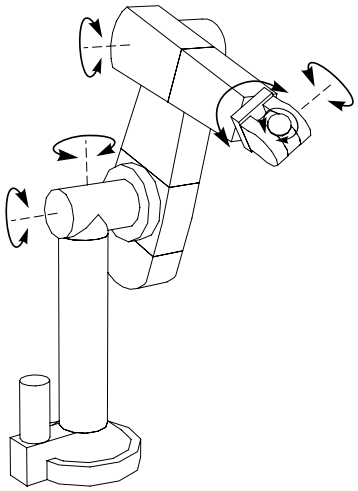

Figure 1: Robotics "armatures" have typically only one-dimensional revolute joints.

2 Robotics armatures

This Section explains the differences between the default approaches towards armatures in Blender and in robotics. ("Kinematic chain" is the more usual name for armatures in robotics.)

2.1 Armature joint types

Figure 1 depicts a "classical" *serial robot* arm, with six revolute joints. Figure 2 depicts the topology of a typical "humanoid" robot: each subset in this tree-structure is a serial robot it itself. Blender offers only *spherical joints*, and requires the programmer to specify *constraints* to "block" some of the degrees of freedom. So, armatures based on real revolute joints have only very ineffcient kinematics calculations in Blender, while these constraints are "built-in" in the usual kinematics algorithms in robotics.

These differences between *spherical joint-based* and *revolute joint-based* armatures have some, but rather limited, consequences for the code that is already in Blender:

- The programmer must be able to assign to each joint in an armature one of the following *types*: *revolute*, *prismatic*, *spherical*, *gimbal*, etc.
- The meaning and construction of the Jacobian matrices and their mathematics as implemented in *iksolver* remain basically the same, but the "constructors" must be aware of the type-dependency of the armature joints.

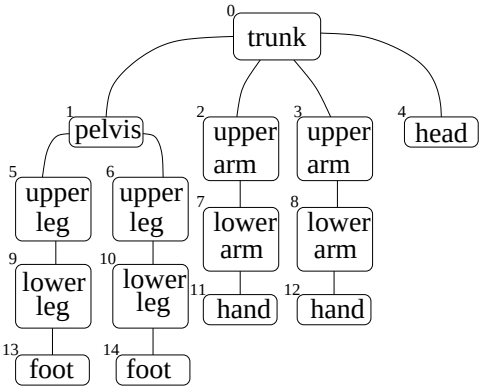

Figure 2: The "topology" of a humanoid robot has the form of a *tree*, with each node possibly being a serial chain in itself.

2.2 Realistic physical properties for armatures

In order to make Blender armatures more appropriate for use in robotics research, an armature data structure should contain the following information, ordered according to usefulness and complexity:

- *Kinematic state*: position, velocity and acceleration, as function of the time.
- *Dynamic state*: mass distribution of the bodies that make up the armature; force (or torque) applied at the joint; elasticity of the joint; and friction in the joint.

  In its simplest form, this information is time-invariant, and given by constant property values. In a more advanced mode, these are function pointers to which the programmer can attach callback functions. This can happen via the built-in Python, or by dynamic linking with an external code library (to obtain the highest efficiency).

Of course, corresponding interfaces must be provided to fill in and monitor this information during programming or during a simulation.

With the dynamics library *ODE*, Blender contains already infrastructure for realistic dynamic simulation, but this is only good for not-connected sets of

bodies: bouncing balls, colliding objects, etc. Armatures can be done with it too, but then each joint introduces lots of "stiff" constraints, and that approach introduces numerical inefficiency and inaccuracies.

3 Constraint-based IPO's

Interpolation curves (IPOs) in Blender are "tuned" via *key frames*: the programmer defines some points where an armature has to pass through, and uses spline-like curves to provide smooth interpolation through these points. This approach is very flexible, in that about any possible motion can be programmed and tuned this way, but it is often too difficult for:

- *Complex motions*, i.e., with under- or over-constrained systems. (An under-constrained system is often called *redundant*.) Apparently, Blender treats constraints as an ordered list of "rules" to apply to the representation of the armature and its motion, but this can be made much more efficient for cases with *linear constraints* (or locally linearizable constraints) on the armature's motion parameters.

- *Natural motions*, i.e., motions that follow the laws of physics. For example, the natural falling, walking or jumping of a human body is governed by the laws of Newtonian dynamics.

For both categories of motions, robotics has developed quite efficient algorithms over the last decade. The following subsections outline what these are and how these could be integrated into Blender.

The easiest way to describe these constraints is to look at the mathematical representations. So, every armature has the following *Jacobian relationship* between, on the one hand, the velocities $\dot{\boldsymbol{q}}$ at the armature's joints, and, on the other hand, the instantaneous velocity ("twist") $\mathbf{t}$ at the "end-point" of the armature:

$$\mathbf{t} = \boldsymbol{J}\dot{\boldsymbol{q}}. \tag{1}$$

$\boldsymbol{J}$ is the matrix whose columns each contain the end-point twist generated by a unit velocity at the corresponding joint; $\mathbf{t}$ is always a six-vector. So, if the armature has more than six degrees of freedom in its joints, Eq. (1) has multiple solutions. Hence, it always has a *null space*, i.e., a set of joint velocities that do not move the end-effector:

$$\text{Null}\left(\boldsymbol{J}(\boldsymbol{q})\right) = \left\{\dot{\boldsymbol{q}}^N \mid \boldsymbol{J}(\boldsymbol{q})\,\dot{\boldsymbol{q}}^N = \mathbf{0}\right\}. \tag{2}$$

This null space depends on the current joint positions. Equation (2) implies that an arbitrary vector of the null space of the Jacobian can be used as an *internal motion* of the robot:

$$\mathbf{t} = \boldsymbol{J}(\boldsymbol{q})\,\dot{\boldsymbol{q}} = \boldsymbol{J}(\boldsymbol{q})\left(\dot{\boldsymbol{q}} + \dot{\boldsymbol{q}}^N\right). \tag{3}$$

The inverse kinematics of a redundant armature require the user to specify a criterion with which to solve the ambiguities in the joint positions and velocities (internal motions) corresponding to the specified end-point position and velocity. Some examples of redundancy resolution criterions are:

1. Keep the joints as close as possible to a specified position. The goal of this criterion is to avoid that joints reach their *mechanical limits*. A simple approach to reach this goal is to attach *virtual springs* to the joints, with the equilibrium position of the springs near the middle of the motion range of the joints. With this spring model, the redundancy resolution criterion corresponds to the minimization of the potential energy in the springs.

2. Minimize the *kinetic energy* of the manipulator, [13].

3. Maximize the *manipulability* of the manipulator, i.e., keep the robot close to the joint positions that give it the best ability to move and/or exert forces in all directions, [9, 15, 18, 21].

4. Minimize the *joint torques* required for the motion. The goal of this criterion is to avoid saturation of the actuators, and to execute the task with minimum "effort," [8, 12].

5. Execute a high priority task but use the redundancy to achieve a lower priority task in parallel, [19].

6. Avoid obstacles in the robot's workspace. For example, a robot with an extra shoulder or elbow joint can reach "around" obstacles, [2, 11].

7. Avoid singularities in the robot kinematics, [1, 3, 16, 20, 22].

8. Travel through singularities while keeping the joint velocities bounded, [5, 14].

Many of these redundancy resolution criterions (implicitly or explicitly) rely on the concept of the *extended Jacobian*, [1]. This approach starts from the observation that the $6 \times n$ Jacobian can be made into

a $n \times n$ matrix by adding $n-6$ rows to it, collected here in a $(n-6) \times n$ matrix $\boldsymbol{A}$:

$$\bar{\boldsymbol{J}} = \left(\begin{array}{c} \boldsymbol{J}(\boldsymbol{q}) \\ \hline \boldsymbol{A}(\boldsymbol{q}) \end{array} \right). \tag{4}$$

This is equivalent to adding $n-6$ *linear constraints* on the joint velocities:

$$\boldsymbol{A}(\boldsymbol{q})\dot{\boldsymbol{q}} = 0. \tag{5}$$

In order to obtain a full-rank extended Jacobian $\bar{\boldsymbol{J}}$, the constraint matrix $\boldsymbol{A}$ must be full rank, and *transversal* (or "*transient*") to the Jacobian $\boldsymbol{J}$, i.e., the null spaces of $\boldsymbol{A}$ and $\boldsymbol{J}$ should have no elements in common, [23]. Equation (4) then has a uniquely defined inverse:

$$\bar{\boldsymbol{J}}^{-1} = \left(\boldsymbol{B} \mid * \right). \tag{6}$$

The $n \times 6$ matrix $\boldsymbol{B}$ is a so-called *generalized inverse*, or *pseudo-inverse*, often denoted by $\boldsymbol{B} = \boldsymbol{J}^{\dagger}$, [4, 6, 17]: it satisfies $\boldsymbol{J}\boldsymbol{B} = \boldsymbol{1}_{6\times 6}$ and $\boldsymbol{B}\boldsymbol{J} = \boldsymbol{1}_{n\times n}$. (This follows straightforwardly from the definition of $\bar{\boldsymbol{J}}$.) With it, the forward velocity kinematics can be "inverted":

$$\dot{\boldsymbol{q}} = \boldsymbol{B}\ \mathbf{t}. \tag{7}$$

Do not forget that the resulting joint velocities depend on the choice of the constraint matrix $\boldsymbol{A}$. The following paragraphs derive this general result of Eq. (7) in more detail and in an alternative way for the particular example of the kinetic energy minimization criterion. The kinetic energy T of a serial armature is of the form

$$T = \frac{1}{2}\dot{\boldsymbol{q}}^T \boldsymbol{M}(\boldsymbol{q})\ \dot{\boldsymbol{q}}. \tag{8}$$

Since T is a *positive scalar* (and hence $T^T = T$), the inertia matrix $\boldsymbol{M}$ is both invertible and symmetric. Minimizing the kinetic energy, while at the same time obeying the inverse kinematics requirement that $\mathbf{t} = \boldsymbol{J}\,\dot{\boldsymbol{q}}$, transforms the solution to the following *constrained optimization problem*:

$$\left\{ \begin{array}{rcl} \min\limits_{\dot{\boldsymbol{q}}} \quad T & = & \frac{1}{2}\dot{\boldsymbol{q}}^T \boldsymbol{M}(\boldsymbol{q})\,\dot{\boldsymbol{q}}, \\ \text{such that} \quad \mathbf{t} & = & \boldsymbol{J}(\boldsymbol{q})\,\dot{\boldsymbol{q}}. \end{array} \right. \tag{9}$$

The classical solution of this kind of problem uses *Lagrange multipliers*, [7, 24], i.e., the constraint in (9) is integrated into the functional T to be minimized as follows:

$$\min_{\dot{\boldsymbol{q}}} \quad T' = \frac{1}{2}\dot{\boldsymbol{q}}^T \boldsymbol{M}\,\dot{\boldsymbol{q}} + \boldsymbol{\lambda}^T \left(\mathbf{t} - \boldsymbol{J}\,\dot{\boldsymbol{q}} \right). \tag{10}$$

(For notational simplicity, we dropped the dependence of $\boldsymbol{M}$ and $\boldsymbol{J}$ on the joint positions $\boldsymbol{q}$.) $\boldsymbol{\lambda}$ is the column vector of the (currently unknown) Lagrange multipliers. They can be physically interpreted as the *impulses* (forces times mass) generated by violating the constraint $\mathbf{t} - \boldsymbol{J}\,\dot{\boldsymbol{q}} = 0$. (Check the physical units!) The Lagrange multipliers are determined together with the desired joint velocities by setting to zero the partial derivatives of the functional T' with respect to the minimization parameter vector $\dot{\boldsymbol{q}}$:

$$\underset{1\times 7}{\dot{\boldsymbol{q}}^T}\ \underset{7\times 7}{\boldsymbol{M}} - \underset{1\times 6}{\boldsymbol{\lambda}^T}\ \underset{6\times 7}{\boldsymbol{J}} = \boldsymbol{0}_{1\times 7}. \tag{11}$$

This gives a set of seven equations, in the seven joint velocities and the six Lagrange multipliers. These Lagrange multipliers can be solved for by post-multiplying Eq. (11) by $\boldsymbol{M}^{-1}\boldsymbol{J}^T$:

$$\dot{\boldsymbol{q}}^T \boldsymbol{J}^T = \boldsymbol{\lambda}^T \left(\boldsymbol{J}\,\boldsymbol{M}^{-1}\boldsymbol{J}^T \right). \tag{12}$$

The left-hand side of this equation equals the transpose of the end-point twist, $(\mathbf{t})^T$, and the matrix triplet on the right-hand side is a square 6×6 matrix that can be inverted (at least if the manipulator is not in a singular configuration). Hence,

$$\boldsymbol{\lambda}^T = (\mathbf{t})^T \left(\boldsymbol{J}\,\boldsymbol{M}^{-1}\boldsymbol{J}^T \right)^{-1}. \tag{13}$$

Equations (11) and (13), and the fact that $\boldsymbol{M}$ is symmetric, yield

$$\dot{\boldsymbol{q}} = \boldsymbol{M}^{-1}\boldsymbol{J}^T \left(\boldsymbol{J}\,\boldsymbol{M}^{-1}\boldsymbol{J}^T \right)^{-1} \mathbf{t} \tag{14}$$

$$= \boldsymbol{J}^{\dagger}_{M^{-1}} \mathbf{t}. \tag{15}$$

$\boldsymbol{J}^{\dagger}_{M^{-1}}$ is a $n \times 6 (n > 6)$ matrix, the so-called *weighted pseudo-inverse* of $\boldsymbol{J}$, with $\boldsymbol{M}^{-1}$ acting as weighting matrix *on the space of joint velocities*, [4, 6, 17]. It is not a good idea to calculate the solution $\dot{\boldsymbol{q}}$ by the straightforward matrix multiplications of Eq. (14); better numerical techniques exist, see e.g. [10].

The redundancy resolution approaches based on an *extended Jacobian* yield only *local* optimality. For example, one minimizes the instantaneous kinetic energy, *not* the kinetic energy over a complete motion. The success of the extended Jacobian approach is due to the fact that analytical solutions exist for *quadratic* cost functions only.

3.1 Interface extensions

The features described in the previous sections require some additions to Blender's interface, especially for the specification of *constraints*. In addition, the *IPO* interface is currently not fully compatible with constraint-based programming, because

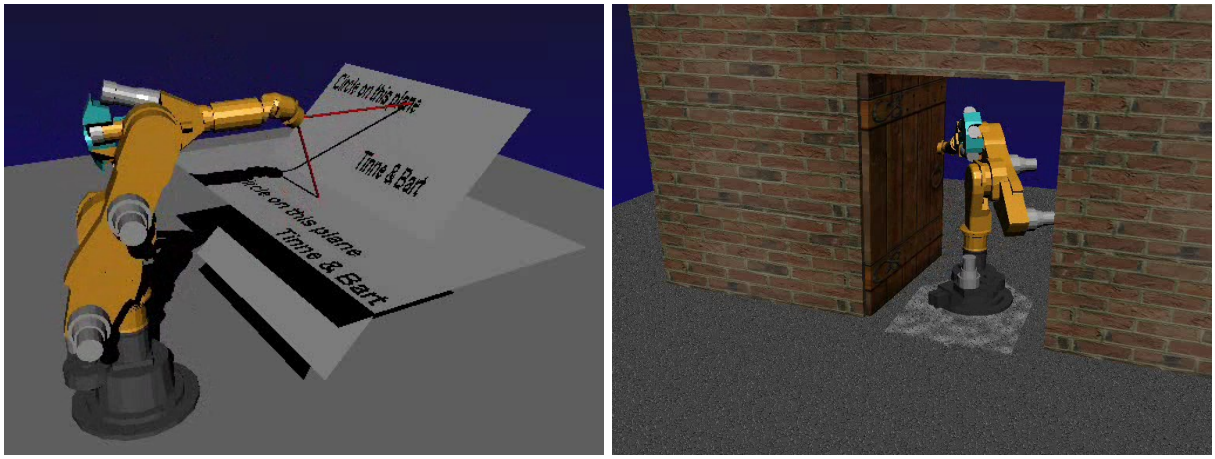

Figure 3: This figure shows snapshots of two simulations rendered in (but not generated by) Blender, and based on the complex motion generation explained in this paper.

the resulting motion need not be splines, and, conversely, changing the resulting curves in the IPO window will most probably violate the constraints that generated the original motion curves.

4 Examples

Figure 3 shows two examples of the automatic constraint-based interpolation technique of Section 3. In both cases, the robot has *eight* degrees of freedom: six revolute joints in the arm, and two driven wheels on the platform.

The first example shows the robot executing two independent motions: each motion involves a simulated "laser beam" traversing a circular trajectory on a plane. The timing and dimensions of both circular motions are arbitrary and not correlated. Each of them gives rise to one linear velocity constraint: the intersection point of the beam with the plane must have an instantaneous velocity tangential to the desired circle.

The second example shows the robot executing a door-opening task. The constraints here are: (i) the mobile part of the robot must move along a straight line through the door, and (ii) the robot rotates around the door's hinge with a prescribed speed. (Note that each of these two motion can be specificied in more than one way.)

5 Conclusions

The extensions to Blender's motion and kinematics libraries discussed in this paper require rather large-scale changes in the source code. However, most of these changes are rather *localized*, and only few new concepts or data structures must be introduced. The author is very interested in brainstorming about the technical contents of this paper, about its relevance to Blender's roadmap, and about the best implementation approach.

Future work of our group will be around the realistic simulation of the dynamics of humanoid robots; or, equivalently, for all biological creatures. This involves not only simulation of motion, but also the introduction of *controllers*, *sensors* and *logic* bricks.

Acknowledgements

Tine Lefebvre, Panagiotis Issaris, Johan Rutgeerts, Tinne De Laet, Friedl de Groote, Bart Demarsin, Diederik Verscheure contributed in one way or another to the ideas and experiments presented in this paper.

References

[1] J. Baillieul. Kinematic programming alternatives for redundant manipulators. In *IEEE Int. Conf. Robotics and Automation*, pages 722–728, St. Louis, MS, 1985.

[2] J. Baillieul. Avoiding obstacles and resolving kinematic redundancy. In *IEEE Int. Conf. Robotics and Automation*, pages 1698–1704, San Fransisco, CA, 1986.

[3] D. R. Baker and C. W. Wampler II. On the inverse kinematics of redundant manipulators. *Int. J. Robotics Research*, 7(2):3–21, 1988.

[4] A. Ben-Israel and T. N. E. Greville. *Generalized Inverses: Theory and Applications*. Robert E. Krieger Publishing Company, Huntington, NY, reprinted edition, 1980.

[5] N. Boland and R. Owens. On the behaviour of robots through singularities. In R. A. Jarvis, editor, *Int. Symp. Industrial Robots*, pages 1122–1134, Sydney, Australia, 1988.

[6] S. L. Campbell and C. D. Meyer, Jr. *Generalized Inverses of Linear Transformations*. Dover, 1991.

[7] R. Courant and D. Hilbert. *Methods of mathematical physics*. Interscience, New York, NY, 1970.

[8] A. S. Deo and I. D. Walker. Minimum effort inverse kinematics for redundant manipulators. *IEEE Trans. Rob. Automation*, 13(5):767–775, 1997.

[9] K. L. Doty, C. Melchiorri, E. M. Schwartz, and C. Bonivento. Robot manipulability. *IEEE Trans. Rob. Automation*, 11(3):462–468, 1995.

[10] G. Golub and C. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, 1989.

[11] J. M. Hollerbach. Optimum kinematic design for a seven degree of freedom manipulator. In Hanafusa and Inoue, editors, *Robotics Research: The Second International Symposium*, pages 215–222. MIT Press, Cambridge, MA, 1985.

[12] J. M. Hollerbach and K. C. Suh. Redundancy resolution of manipulators through torque optimization. In *IEEE Int. Conf. Robotics and Automation*, pages 1016–1021, St. Louis, MS, 1985.

[13] O. Khatib. *Commande dynamique dans l'espace opérationnel des robots manipulateurs en présence d'obstacles*. PhD thesis, Ecole Nationale Supérieure de l'Aéronautique et de l'Espace, Toulouse, 1980.

[14] J. Kieffer. Differential analysis of bifurcations and isolated singularities for robots and mechanisms. *IEEE Trans. Rob. Automation*, 10(1):1–10, 1994.

[15] C. Klein and B. Blaho. Dexterity measures for the design and control of kinematically redundant manipulators. *Int. J. Robotics Research*, 6(2):72–83, 1987.

[16] J. Lloyd. *Robot Trajectory Generation for Paths with Kinematic Singularities*. PhD thesis, McGill University, Montreal, Canada, 1995.

[17] M. L. Moe. Kinematics and rate control of the Rancho arm. In *Proceedings of the First CISM-IFToMM Symposium on Theory and Practice of Robots and Manipulators*, pages 253–272, Wien, Austria, 1973. Springer Verlag.

[18] Y. Nakamura. *Advanced robotics: redundancy and optimization*. Addison-Wesley, Reading, MA, 1991.

[19] Y. Nakamura, H. Hanafusa, and T. Yoshikawa. Task-priority based redundancy control of robot manipulators. *Int. J. Robotics Research*, 6(2):3–15, 1987.

[20] K. A. O'Neil, Y.-C. Chen, and J. Seng. Removing singularities of resolved motion rate control of mechanisms, including self-motion. *IEEE Trans. Rob. Automation*, 13(5):741–751, 1997.

[21] F. C. Park and J. W. Kim. Kinematic manipulability of closed chains. In J. Lenarčič and V. Parenti-Castelli, editors, *Recent Advances in Robot Kinematics*, pages 99–108, Portorož-Bernardin, Slovenia, 1996. Kluwer.

[22] T. Shamir. The singularities of redundant robot arms. *Int. J. Robotics Research*, 9(1):113–121, 1990.

[23] T. Shamir and Y. Yomdin. Repeatability of redundant manipulators: Mathematical solution of the problem. *IEEE Trans. Autom. Control*, 33(11):1004–1009, 1988.

[24] G. Strang. *Introduction to applied mathematics*. Wellesley-Cambridge Press, Wellesley, MA, 1986.