Chapter 3

Dynamics of the Grand Piano Action

3.1 Introduction

In this chapter, I present a dynamical model of the grand piano action. The intended use of this model is for interactive simulation with haptic display: to re-create, with a motorized keyboard, the *touch response* of the piano. Additionally, the model will be used to enable a synthesizer to re-create the *sound response* of the piano by facilitating a proper mapping from input gesture to sound parameters (the hammer strike velocity and strike time) —where 'proper' is taken to mean reflective of the behavior of the grand piano action. The employment of the model in an interactive simulator and its role as a *virtual* piano action will be detailed in the next chapter. In the present chapter, I concentrate on the development of the model and its expression as an impedance operator, suitable for later use in emulating touch response, and as a mapping from gesture to sound parameters, suitable for use in emulating the sound response.

There exist some significant features in the behavior of the piano action which we are interested in capturing in our dynamical model. These features effectively modulate, as a function of key depression and depression rate, the otherwise configuration independent inertial, dissipative and gravity-balance forces which are felt by a finger pushing on the key. I will highlight three such features:

- The piano action relies on an escapement or 'trip' mechanism for its operation. The jack, which initially propels the hammer toward the string by pushing on the hammer knuckle, is pivoted out from under the hammer knuckle just before hammer/string impact in a process called 'letoff'. The hammer will subsequently rebound off of the string, to be caught by the repetition lever or backcheck rather than the jack. Associated with letoff, but only noticeable at low key depression rates, is a period of increased response force just before the key hits the keybed. This brief rise in reaction force is called 'letoff resistance'.

- After a complete key depression, a repetition mechanism (sometimes called the 'double escapement mechanism') facilitates a reset of the jack under the hammer knuckle when the key is allowed to rise off the keybed by a short distance. The hammer is then ready for a repeat strike, and the letoff resistance will once again be encountered if the key is depressed slowly.
- Using only shallow depressions, the hammer may be bounced on the jack and the response forces from the key will suggest to the player that a bouncing object is being manipulated through the key.

Other features of the piano action behavior which will be of interest for emulation include the dependence of the manner in which the key returns to rest position on the manner in which it was hit and released.

To attempt to capture these myriad behaviors in a model presupposes a model form which can exhibit the effects of making and breaking contact between bodies —for at the heart of an escapement mechanism is the making and breaking of contact between system bodies, or what might be called 'changes in kinematic constraint'. [1] To change the operative constraint is the end goal of an escapement mechanism and furthermore, a change in constraint is usually the means of activating an escapement. The fact that changing kinematic constraints play such a large role in the operation and in the mechanical impedance of the piano has greatly influenced our choices regarding modeling approach and simulator architecture. For example, we have decided against system identification or 'black-box' modeling methods. We expect that methods which produce linear models will perform poorly when asked to describe the behavior of the piano action, owing to the discontinuities inherent in its changing constraints. We have decided instead to pursue a multibody dynamical model, parameterized according to properties which can be drawn from the system components individually, such as mass properties, damping and spring coefficients, and dimensions. We use the entire behavior of the piano action, including the motion of all of the bodies, to direct our model construction, rather than just the manifest mechanical impedance and hammer-strike parameters.

[1]Some authors refer to 'changing kinematic constraint' using the term 'change in topology' [35], [102], others as 'constraint addition-deletion' [44]. Some authors describe systems subject to changing constraints with the term 'intermittent motion' [103]. The term 'imposition and relaxation of constraints' has also been used in this context [28]. The Russian literature uses 'variable structure'. 'Variable connectivity' is also used.

Our model takes on a special form in order to capture the behaviors which arise from, and are embodied in, these changing kinematic constraints. The model is actually composed of a set of submodels, each of which describe the piano action in one of its constraint conditions. Accordingly, our simulator has been specially designed for simulating multibody systems with changing kinematic constraints. The simulation of a complete escapement is accomplished by simulating through a sequence of submodels, passing the final conditions of each submodel on as initial conditions for the next submodel at the transition times. In our simulator, the sequencing of the submodels actually takes place interactively, at run-time. A finite state machine (fully defined below) is employed to sequence through the submodels as a function of driving input from the user. Essentially, the finite state machine manages the transition times and the transitions themselves (/ie which submodel takes over from the current submodel).

Section 3.2 below includes a survey of models of the piano action which have appeared in the literature. To prepare for the presentation of our piano action model, section 3.3 reviews the various forms in which a general dynamical model involving constraints may be expressed, paying particular attention to the advantages of each form as regards computational efficiency and ease of handling changing kinematic constraints. Section 3.4 will discuss and defend the particular form which we have chosen for our model and introduce the finite state machine. Each of the submodel components which will be put into service by that finite state machine will be defined. Finally, section 3.5 will present our model of the grand piano action, one submodel at a time. Sections 3.6 and 3.7 present some simulation and experimental results which have been used to develop and verify this piano action model.

3.2 Literature Review: Piano Action Modeling

The small number of analytical investigations into the kinematics and dynamics of the piano action which have appeared in the literature will be examined in this section. Except for the omission of models which remain behind the doors of piano manufacturers such as Steinway, Baldwin, Yamaha, or action manufacturers such as Renner and others, and for the omission of occasional treatments possibly appearing in the numerous piano action patents, the following may be considered a reasonably comprehensive review of analytical investigations into the dynamics of the piano action.

This section will also review the literature pertaining to finite state machines and will present a broad outline of the major forms in which a dynamical model may be expressed.

3.2.1 Dynamical Models of the Piano Action

Walter Pfeiffer published a set of books which treat various aspects of the grand and upright piano actions. Pfeiffer's interest was in uncovering possible design improvements. His book, Whippen and Hammer, deals primarily with the kinematics of sliding contact between the capstan screw and leather-covered whippen heel with gear theory. [81]. Dynamics of the piano action were considered in [80] and [79] using elementary models and applications of conservation of energy.

Dijksterhuis developed a simplified dynamical model of the piano action in [27] (also quoted in [97] and [98]) which accounted for the inertia of the hammer, whippen, jack, and key with an equivalent mass at the point of force application on the key by considering the mechanical advantage of the key over each of these bodies. The friction and gravity forces at the key were modeled as a constant (configuration independent) force. Dijksterhuis reported an equivalent mass of the key, jack, whippen and hammer of 208 grams at the point of application of a playing force on the key. Modulation of the reaction force by changing kinematic constraints was not considered.

Topper and Wills presented a simple model of the piano action in [93]. The action was modeled as a linearized double mass, wherein the key and hammer were modeled as masses coupled by a spring. Model parameters were estimated by experiment and also varied to 'calibrate' the simulated behavior to experimental behavior. Once again, changing kinematic constraints were not considered.

Guido Van den Berghe presented a detailed model of the piano action in [98]. The complete model description can be found in [97]. Van den Berghe's aims were to account for the mapping from gesture to sound parameters with a model which can be simulated in real-time in a synthesizer. As discussed in Chapter 2 of this thesis, and even more thoroughly by Van den Berghe, an implementation of the proper mapping would amount to a vast improvement over the standard 'velocity sensitivity' of today's synthesizers. Van den Berghe used Bond Graph techniques [55] and the mechanical system simulation package DYNAST. Van den Berghe's model does indeed account for changing kinematic constraints. However, his model was not particularly computationally efficient because it belongs to the class of 'coupled force balance' models (defined below). Basically, springs and/or dampers are placed between each massive body, and the Newton-Euler equations are applied for each mass-center, resulting in a model with many degrees of freedom. Van den Berghe reports a 4 hour simulation time on a 486DX33 PC for 1 second of real time. A reduced (linearized) model runs 15 seconds on a DEC 5000/33 for 1 second of real time.

3.2.2 Applications of the Finite State Machine

Finite state machines have a long history of application in design and control. After all, the digital computer is itself a finite state machine, each of its states being determined by the previous state and sensed input. Finite state machines have been used to control manufacturing systems and design fault tolerant event detection and response systems for many years. Recently, the finite state machine has found application in robot control. Schneider [90] applied state table programming techniques to draft out the behavior of a robot in response to events which were detected during that behavior. Thus the finite state machine was used to manage real-time interactions with the robot's environment. The finite state machine also provided a convenient way of integrating high-level user commands, given the intuitive stimulus/event model. Roger Brockett has made use of finite state machines in [14] to develop position and general language-based motion control strategies for robots. Finite state machines find many applications in walking and juggling robots. See work by Raibert [83] and Bühler and Koditscheck [16]. More recently, Hyde and Tremblay et al. [49] and Tremblay and Cutkosky [94] have applied state table programming techniques to the control of robotic hands.

The application of a finite state machine to the handling of changing kinematic constraints in a mechanical system simulation has been suggested as an extension to the PODE formalism in Chapter 14 of Barzel's book [9], which is further reviewed below.

[56]

3.3 Model Form Overview

Due to the importance of real-time simulation within our project goals, we desire a model expression which is computationally efficient, yet still captures the intricate behavior of the piano action. As stated earlier, we are interested in modeling the changing kinematic constraints which give function to the piano action and have an important effect on the mechanical impedance of the piano at the keys. We shall build a multibody dynamical model, modeling each of the wooden elements as rigid bodies, and interspersing lumped parameter springs and dampers to account for the spring-wire, felt and leather components. Having chosen to use a rigid-body model, and given that most of the constraints we wish to model are unilateral (may act to prevent interpenetration of bodies, but not to hold the bodies together), changing kinematic constraints must be handled carefully (see [36] for a thorough discussion). The addition of a kinematic constraint may arise when a collision (typically detected by an interference checker in simulation) occurs between body boundaries. A deletion of a kinematic constraint, however, may happen in one of three ways: the constraint breaks because of insufficient closing force (unilateral condition), the constraint breaks due to relative sliding (one body slides off of the edge of another), or the constraint breaks due to the restitution of an impact. The impact restitution problem must be handled upon collision detection since the bodies have been assumed rigid and the time intervals over which the impact forces will act may be very short, giving rise to discontinuous jumps in certain velocities. The integration routine must be stopped upon the detection of a collision, the momentum-impulse equations solved for the subsequent velocities, and the integration routine re-started. This issue is not a consequence of discrete simulation, but an integral part of analysis using rigid body models.

A multibody dynamical model, embodied in the equations of motion, may be expressed in one of several standard forms. Below I will describe three major forms: the coupled force balance formulation, the dependent coordinate formulation, and the independent coordinate formulation. These various forms may be expressed sometimes as ordinary differential equations (ODEs), and sometimes as differential algebraic equations (DAEs), as explained below. A simulator can be based on each form, though each model expression carries its own advantages with regard to ease of handling changing constraints and computational efficiency. Other factors worth weighing include ease of model construction, ease of implementation in a simulator, availability of algorithms with good numerical properties, and so on. It can also be said that each major modeling technique (Newton-Euler, Lagrangian, or Kane's Method), tends to produce a model in a particular form. The present review, however, will concentrate on model form or expression rather than the formulation process. In general, when the predicted behavior is the same, one form may be translated into another, though when the models are complex, translation is an arduous process.

We have chosen to express our model in an independent coordinate formulation. To provide background for our choice in model form and modeling technique, a few comments about each of the major forms are in order. Brief comments will be made for each form as regards computational efficiency and the manner in which changing kinematic constraints are typically handled.

This small review is not intended to be complete. A comparative study of each of the available model formulations and their associated simulation architectures with regard to their advantages for real-time simulation with haptic display is too large a project to be undertaken here. Contributions to this area have been made by researchers in many fields including computer graphics, numerical methods, robotics, and of course dynamics. Furthermore, interest in real-time and interactive simulation is on the rise, spurned by the digital computer's continuing gains in computing power. Texts with overviews on the field of dynamic simulation are available from the dynamics community. See [43] for a review of numerical methods for real-time mechanical system simulation and [32] for a discussion of both model formulation and simulation methods focused on real-time applications. From the computer graphics community, see [9]. Review papers include [25], which addresses some issues in haptic display. The following outline is drawn from the papers cited therein and each of the above texts (especially [32]) and this outline is unique only in that it carries a broader perspective than any single available review.

For reference in the following discussion, Figure 3.1 is presented.

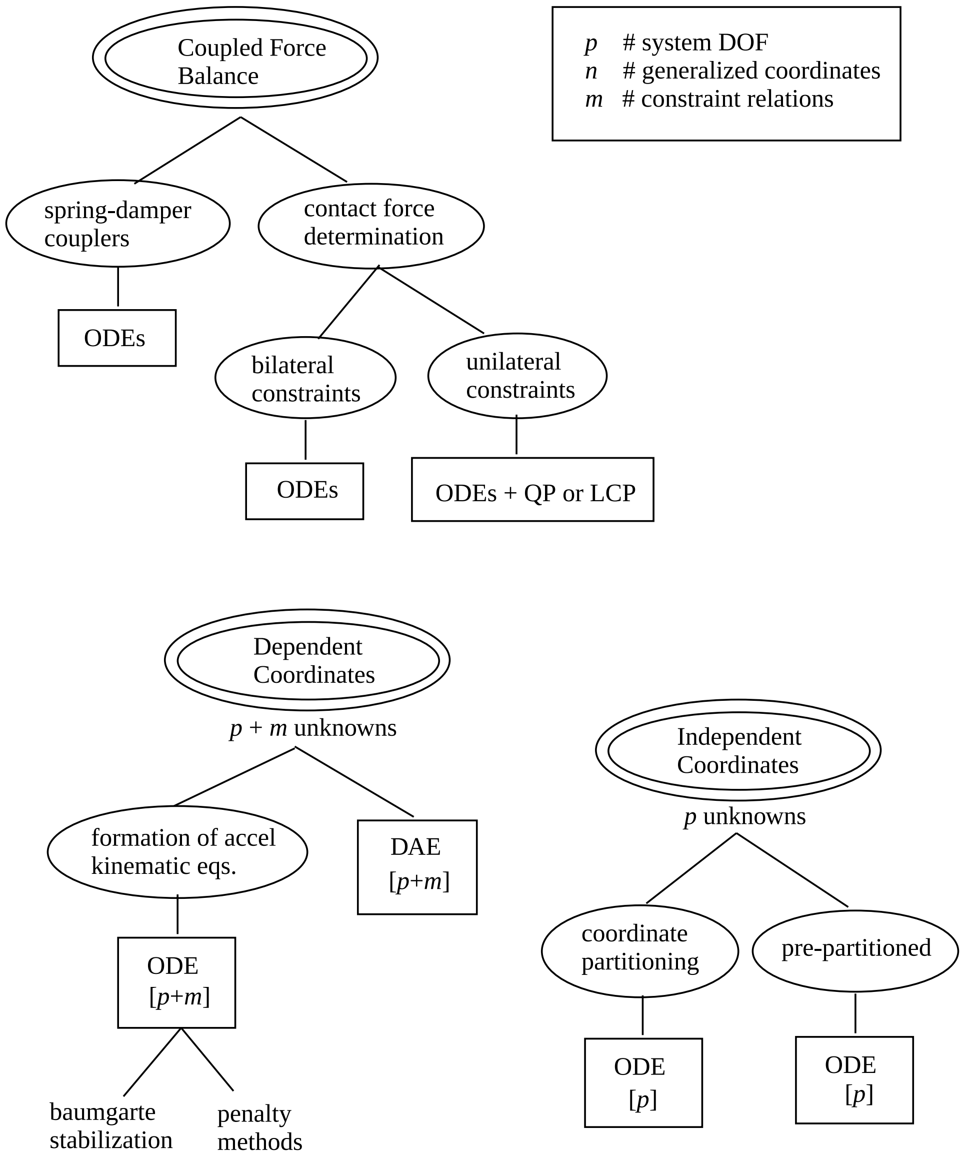


Figure 3.1: *Model Formulations*

3.3.1 Coupled Force Balance Formulations

The term 'Coupled Force Balance' refers to those methods which formulate independent equations of motion for each subsystem. A subsystem is defined as a particle, body, or collection of bodies which within itself is not subject to changing constraints. Changing constraints arise between subsystems [2].

Thus a set of differential equations (models) is produced, one for each subsystem, which are completely independent from one another. Constraint equations relating the motion or configuration of these subsystems are not used to couple subsystem models or formulate system-wide models. Instead, when an interference checker (running in parallel with the dynamic simulation) detects a surface contact between the boundaries of one subsystem and another, an appropriate interaction force is computed in one of two ways (outlined below) and communicated to each of the subsystems for use in their respective (independently running) forward dynamics simulations. The distinguishing factor of a Coupled Force Balance formulation is that, where a constraint condition is subject to change, no constraint relation is employed.

Coupling through Spring-Damper Pairs

The simpler of the two methods for imposing the (possibly changing) 'constraints' in the coupled force balance scheme consists of coupling subsystems through intervening springs or spring-damper pairs when the bodies make contact with one another. In some analyses, these spring-damper pairs are called 'impact pairs' [36] (and references therein). During those times interference between two subsystems is detected, an interaction force is communicated to each of them (for use in their balance) according to the constituent equations of the intervening spring-damper element pair and Newton's third law. To mimic unilateral constraints, these spring-damper pairs are only allowed to exert repulsive forces on the two bodies. When a change from compressive to tensile force is detected, the spring-damper pair is removed. This method obviously produces models with more degrees of freedom than necessary (unless the coupling between bodies really *is* best modeled as compliant or damped), since with the incorporation of such an intervening element, the number of degrees of freedom is not reduced; no constraint equations are imposed. Many examples of this approach may be drawn from the computer graphics community: See Moore and Wilhelms [76]. From the computer music/graphics community, see the work of Cadoz, Luciani, and Florens [20], [63], [30]. Platt and

[2]To derive the equations of motion for each subsystem, various methods are used, including those associated with the dependent coordinate or independent coordinate formulations discussed below (though, as stated above, the 'changing constraints' are not used in the subsystem model formulations). A number of researchers in the computer graphics community, however, consider only particles, or perhaps independent rigid bodies as subsystems. Those researchers which consider only particles apply Newton's second law separately to produce the force balances [20], [63], [30]. When bodies are considered, Newton-Euler principles are used to produce the force balances.

Barr [82] extend these methods to non-rigid body dynamics by applying methods of constrained optimization.

From a rigid-body modeling standpoint, this method may be regarded as approximate since the bodies may in fact interpenetrate slightly. Certain authors therefore call this method non-analytic [6], [7]. Within the computer graphics literature, the term 'penalty method' is generally used to refer to the use of spring-damper pairs to impose (possibly unilateral) constraints. (I find this a somewhat misleading practice, since 'penalty method' is already used to refer to a technique for the stabilization of numerical methods —though certainly the two methods are very closely related).

One advantage of the spring-damper pair is that impulsive forces (forces acting over infinitesimal time intervals) do not arise. The compliant coupler naturally acts to smooth the interaction force between bodies. If the value of the stiffness coefficient of the couplers is increased in an attempt to approximate rigid body behavior, however, the system equations may become 'stiff' and numerically ill-conditioned. A 'stiff' system is one whose dynamical differential equations possess a solution with widely disparate (or widely varying) time constants.

Repeated Impulse

Another method for determining the interaction force between contacting subsystems may be considered at this juncture —it may also be considered 'non-analytic'. This method is not represented in Figure 3.1. Hahn [38] suggests modeling contact forces strictly with impulses. When collisions occur, the time interval for subsystem interaction is assumed to be very short, thus impact forces are involved, giving rise to discontinuities in velocities. (Note that the velocity of a body which is connected, but remotely located to the impact location may change discontinuously). The resulting change in velocities may be found by application of the impulse-momentum equations. A proper treatment must consider the entire system (both subsystems). To handle resting contact (maintained contact), Hahn simply assumes repeated impacts, with high repetition rate. Mirtich and Canny have extended the repeated impulse technique [73] [74] and also combined it with constraint techniques [72].

Contact Force Computation

The second method for imposing the possibly changing constraints between subsystems in the coupled force balance scheme involves computing the interaction forces for subsequent use in the force balances of the contacting subsystems from an inverse dynamics model formulation. This method may be further broken down by whether the constraint condition imposed is bilateral or unilateral.

Bilateral Constraints

By assuming that a bilateral constraint is immediately locked into place when two bodies make contact, an inverse dynamics model may be set up and used to solve for the interaction forces. That is, since the kinematic state is known (relative acceleration between subsystems assumed nil), the interaction forces may be determined. This results in a linear system of equations. The entire system is once again involved in this equation, for each contact point is considered. Singular value decomposition techniques are recommended for robustness near singular configurations [10]. This technique is used by Barzel [10] and Isaacs and Cohen (without treating closed kinematic chains) [51] for application in interactive computer graphics. Both of these authors further exploit the inverse dynamics problem formulation to allow a user-animator to specify desired motion of an object or character. The specified accelerations are used to solve for the forces which would produce that motion. Assembly of subsystems into objects of coupled subsystems may be accomplished by requesting critically damped approach velocities as joints and other 'constraints' are instantiated [10].

Unilateral Constraints

The interaction forces between contacting subsystems may also be determined by solving the inverse dynamics problem while imposing inequality conditions on certain variables. Specifically, non-interpenetration of contacting surfaces and repulsive contact forces (unilateral 'constraints') are stipulated. Denoting the relative normal acceleration between contact points by a_i and the interaction force as f_i, these two conditions can be written:

$$a_i \geq 0, \quad f_i \geq 0 \tag{3.1}$$

where i indexes all contact points. A third condition results from the stipulation that the forces be conservative:

$$f_i a_i = 0 \tag{3.2}$$

In words, this last equation means that, if the interaction force is nonzero, the relative acceleration must be zero (resting contact), else if the force is zero, the acceleration must be positive, in which case the bodies are moving apart. The full system dynamics may be formulated as a linear relation between a vector **a** of contact point accelerations and a vector **f** of interaction forces,

$$a = Af + b \tag{3.3}$$

where A (containing the masses and contact geometries) is symmetric and PSD and b (containing the unknown interaction and inertial forces) is in the column space of A. The above inequality conditions may combined with the system dynamics to formulate a Quadratic Programming (QP) problem (see [6]) or a Linear Complementarity Problem (LCP) (see [7]).

Treatment of the colliding subsystems as rigid bodies necessitates the resolution of impulse forces, using once again the inequalities noted in Equation 3.1. Upon collision, the simulation is stopped and the impulse-momentum relations are used to 'resolve' the impulse forces (determine subsequent system velocities) by solving a QP or LCP. If, after impulse resolution, the subsystems are not accelerating away from one another, they are said to be in resting contact, and the contact forces are found with the solution, once again, of either a QP or LCP.

Lötstedt provides a thorough derivation of the LCP for the impulse resolution and contact force determination problems, also noting that it can be stated as a QP [61]. These unilateral contact force computation methods have been introduced to the computer graphics community by Baraff [6]. Treatments of friction between contacting bodies are presented by Lötstedt in [62] and Baraff in [7]. Lee, Ruspini and Khatib have recently applied the methods of Baraff, citing [6], to robotic simulation in [57].

More comments on the Coupled Force Balance Formulation

With the advent of object-oriented programming techniques, the coupled force balance modeling approach is receiving a fair amount of attention since it fits so naturally into the object-oriented prescript. Most importantly from our viewpoint, changing kinematic constraints are handled quite easily in models expressed as coupled force balances. A communication line between objects (over which the interaction force is relayed to each force balance equation) is simply toggled on and off when interference or clearance is detected. Resolution of impulses may be performed upon detection of a collision. Of course one of the largest challenges in multibody dynamics simulation is the detection of interference and the determination of contact points when two bodies collide (called the collision detection problem), for this determines the points of application of the interaction forces. So long as an effective interference checker is used, the coupled force balance form may easily accommodate points of force application which are not known ahead of run-time, for no constraint equations are formulated.

The number of second order differential equations to be integrated in the coupled force balance scheme equals the number of independent coordinates used. There are no dependent coordinates in this scheme, since direct coupling is effectively eliminated by placing a spring or spring-damper pair between all masses. Accordingly, the number of independent coordinates (degrees of freedom) is large (compared to a formulation involving constraints), and computational efficiency must be regarded as poor.

Noteworthy for this project and its setting in the field of computer music is the fact that the methods of both Cadoz [20], [63], [30] and Van den Berghe [98] fall under the umbrella of coupled force balances, and reportedly make high demands (from our viewpoint) on computational hardware.

3.3.2 Dependent Coordinate Formulations

For the following discussion, it will be necessary to carefully define a few quantities.

The number n of generalized coordinates for a system of bodies S in a reference frame A is the smallest number of scalar quantities such that to every assignment of values to these quantities and the time t there corresponds a definite admissible configuration of S in A (see [54] p. 39). If restrictions are imposed on the positions or orientations which S may occupy, S is said to be subject to configuration constraints, expressed as holonomic constraint equations. When a system S is subject only to configuration constraints, then S is said to be a holonomic system possessing n degrees of freedom in A. Note that for holonomic systems, the number of degrees of freedom, p, is equal to the number of generalized coordinates, n.

If restrictions are imposed on the *motions* of S, then S is said to be subject to motion constraints, expressed as nonholonomic constraint equations. Nonholonomic constraint equations may *also* arise if, in constructing a model, one chooses to use more generalized coordinates than exist degrees of freedom for the model. Then m constraint equations are written to express the m dependent coordinates in terms of the p independent coordinates. The integer m is given by

$$m = n - p \tag{3.4}$$

A model formulation in which the dependent coordinates are treated as unknowns along with the independent coordinates is called a Dependent Coordinate Formulation. Dynamical models in dependent coordinates are often produced using Lagrangian methods. For example, the method of Lagrange multipliers entails *adjoining* the constraints to the n dynamical differential equations resulting in a set of n equations in $(n+m)$ unknowns. Adjoining entails appending the jacobian of the constraint matrix with a pre-multiplying vector of m undetermined coefficients (Lagrange multipliers) to the n Lagrange equations. The m constraint equations themselves may then be used together with the n dynamical equations to bring the number of equations up to the number of unknowns in one of two ways. Firstly, the algebraic constraint equations may be used directly with the differential equations if a Differential Algebraic Equation (DAE) solver is available. DAE solvers are not the most numerically efficient and are not free from stability problems [43]. The second manner in which the constraint equations may be incorporated is by differentiating them twice (in the case of configuration constraints) or once (in the case of motion constraints) to produce m *acceleration* constraint equations which may be integrated along with the dynamical differential equations (because they are now of the same order) in an ODE solver. Due to their derivation through two differentiations (in the case of configuration constraints), the differential constraint equations are unstable, and will require special treatment during integration (see [32], p. 162). Treatments include Baumgarte stabilization [11] and Penalty methods. Baumgarte stabilization essentially attaches the solution via a virtual spring and damper to the manifold of the constraint equations. In the Penalty formulation, the constraint equations are once again incorporated into the dynamical problem directly, penalized by a large factor. Gradient feedback methods to 'constrain' the energy in the simulated system are also available [110]. See Yen, Haug, and Tak [109] for a method to convert DAEs to ODEs on manifolds, which may be used to some advantage.

DAE simulators are favored in the real-time flight simulation and automobile simulation communities, where changing kinematic constraints are occasionally of interest. See, for example [43].

Changing kinematic constraints are handled rather conveniently in the dependent coordinate formulation, because only the adjoined algebraic equations need be swapped out at the transition times. The dynamical differential equations and their state variables continue unaltered. Gilmore and Cipra cover the simulation of planar systems in dependent coordinates with changing kinematic constraints in the two-part paper [35] and [36]. The m constraint equations are automatically (using an 'incidence matrix' containing the continually updated system topology) swapped in and out of the sparse matrix formulation in which the $n+m$ equations have been lined up. The impulse-momentum principle is used to solve for the post impact velocities upon collision detection. Based on the impact response, the post-impact topology is determined (a new constraint may or may not be added).

Haug, Wu, and Yang formulate a model in dependent coordinates in the three-part paper [44], [106] [105] in which changing constraints, impact, and friction are treated in one framework. The separability of the constraint equations from the dynamical equations is used to advantage.

3.3.3 Independent Coordinate Formulations

A model in independent coordinates has only as may dynamical differential equations as there exist degrees of freedom for that model. (Note that some authors call this formulation 'reduced' [103] and [101])

Although it is possible to formulate the Lagrange equations in independent coordinates [103], I will use Kane's equations (which include the notion of generalized speeds) in the following discussion. Kane's method naturally produces models in the independent coordinate formulation.

Generalized speeds are defined by equations of the form

$$u_r = \sum_{s=1}^{p} Y_{rs}\dot{q}_s + Z_r, \quad (r = 1, ...n) \tag{3.5}$$

where Y_{rs} and Z_r are functions of $q_1, ...q_n$ and possibly time t, but not of $u_1, ...u_n$.

The nonholonomic constraint equations active in the system may be used to eliminate the dependent generalized coordinates to produce a formulation in only p independent coordinates, with one proviso: the nonholonomic constraint equations must be either holonomic or 'simple' nonholonomic constraints. A simple nonholonomic constraint is expressible by a relationship between the generalized speeds u_i, $(i = 1, ...n)$ in the following form:

$$u_r = \sum_{s=1}^{p} A_{rs}u_s + B_r, \quad (r = p+1, ...n) \tag{3.6}$$

where A_{rs} and B_r are functions of $q_1, ...q_n$ and possibly time t, but not of $u_1, ...u_n$.

The dependent coordinates are found during integration either by solving the position problem at each time step, or, more conveniently, by integrating the constraint equations 3.6, which are only first order and stable, along with the differential equations.

Changing kinematic constraints are not so easily handled within the independent coordinate formulation. When the constraint equations change, the entire model must be reformulated; a new set of independent coordinates must be found. Integration of the equations of motion must be stopped, the equations swapped out, and re-started at each change of kinematic constraint.

A method for automatically handling the changing kinematic constraints when the generalized coordinate definitions themselves do not change, but the subset of coordinates which may be considered independent from among the entire set does change, has been proposed by Wehage and Haug [103]. The method is called coordinate partitioning. The jacobian matrix of constraint relations may be solved for the independent rows at each time step, and used to direct and maintain a set of well-conditioned (maximally independent) coordinates.

For handling the impulse-momentum problem in a formulation congruous with Kane's equations (Generalized Impulse, Generalized Momentum), the techniques of Djerassi are available [28].

3.3.4 Closing comments on Overview

This overview suggests that among these broadly categorized model forms, there exists a tradeoff between computational efficiency and ease of accommodating changing constraints. The spring-damper coupler method is simple to implement, may be easily managed for handling changing kinematic constraints without stopping to reformulate model, swap models, or even compute impulses, but it is quite computationally intensive. On the other end of the spectrum are the independent coordinate model formulations which do not easily handle changing constraints, but are maximally computationally efficient.

The computer graphics community has shown the most interest in simulation of changing kinematic constraints, with their interest in animating characters which behave in complex real-world environments. Unfortunately, the model formulation methods of the computer graphics community are generally not very sophisticated. By contrast, attention on changing constraints from traditional dynamicists has been small. There seems to have existed some reluctance to incorporate a constraint into a model if it is subject to change, since its change will of course render the model useless and require re-construction from scratch. Said another way, to incorporate the changing constraint into a model necessitates that such model must be considered transient, and its applicability be continually checked with an inequality condition. With the emergence of computer-aided model formulation, however, model construction need no longer be considered a chore. The checking of inequalities is easily managed by a simulation routine.

An inequality is not so easily handled in the construction of a model, or even in the direct use of a model, as in the inverse dynamics with unilateral conditions as promoted by Baraff [6]. Either quadratic programming or linear complementarity problems arise. To this author, it seems that the inequalities are better handled by the simulation algorithm than by the modeling method or model formulation.

3.4 Our chosen modeling method

We have chosen to construct our piano action model in the independent coordinate formulation, expressed as an ODE, for its computational efficiency and ease of implementation in a simulator, and taken the viewpoint that the difficulty in handling changing kinematic constraints with ODEs is an opportunity to contribute rather than a liablity. Having chosen the independent coordinate formulation and the ODE, we have at our disposal the most standard numerical methods for simulation, leaving room for attention to the relatively complex issue of changing kinematic constraints.

The other influencing factor on our decision of model form was the availability, early on in the project, of an efficient modeling method which naturally produces models in the independent coordinate formulation (Kane's method) and an associated software package for streamlined formulation of the model: AUTOLEV [89]. [3]

[3] I have concentrated my comments above on the model form or expression rather than the methods for construction of the model, but given that the five body piano action is, quite undeniably, a complex system, the choice in modeling technique and associated software package is a very important one. A technique which encourages enough *divide* as

There is another important factor which makes our piano action system ammenable to modeling with independent coordinates, despite its changing kinematic constraints. That is the fact that each of the constraint conditions can be formulated at the outset, allaying the need to run a complex collision detector and constraint formulator during simulation. While the piano action does exhibit changing constraints, it is unlike general simulation of bodies in 3D space (which concerns the computer graphics community) because the number of different possible states is comparatively modest. It is possible to parameterize the point of first contact between bodies with dependent generalized coordinates which are kept up to date during simulation due to the fact that each of the bodies in the piano action are either pivoted to ground or are pivoted to a body which in turn is pivoted to ground. There will be no need to stop and reformulate the equations of motion and constraint equations each time the constraint conditions change. All submodel ODEs (one for each constraint condition) can be formulated ahead of run-time. The set of coordinates will not change, simply the enforcement of the various possible constraint conditions will change.

To account for the changing kinematic constraints with the ODE form, submodels (each a separate ODE) are linked together to form a full ODE which may be said to be only piece-wise continuous. Discontinuities are allowed in both the specification and in the solution (or simulated state trajectory) of the full ODE at the transition times. Our simulator is specially designed to accommodate these piece-wise continuous ODEs. Basically, we wrap a standard ODE solver in an algorithm which can locate events during the solution and manage the exchange of submodels in and out of the solver, keeping the relevant submodel in place and starting each with the proper initial conditions.

Piece-wise continuous ODEs and their use in realizing changing kinematic constraints have been discussed by Barzel in [9], among others. Barzel used the abbreviation PODE for piece-wise continuous ODE. Here I will introduce our extension to Barzel's PODE formalism which I will rather boldly call Event Processing Interactively Sequencing Ordinary Differential Equations (EPISODEs). Whereas the actual order in which the submodels are taken on is pre-determined in the PODE, the order of the sequence is not determined until run-time in the EPISODEs. Only the set of submodel ODEs from which the selections are made in real-time is predetermined in an EPISODE. To decide which, from among the set of all submodels, is to be the next submodel for simulation, (to manage

well as *conquer* is to be preferred. The responsibility for certain modeling decisions should be kept in the hands of the analyst while the drudgery of algebraic manipulation is alleviated. Some of the more powerful modeling software packages available today were not designed in this spirit. It can be argued that some dynamical modeling tools take too much responsibility off the hands of the user-analyist, decreasing net effectiveness.

The Bond Graph methods employed by Ven den Berghe likewise encourage a divide and conquer approach. However, Bond Graphs [55] are rather unwieldy in two dimensions, since they are based on power relations (effort times flow, or force times velocity) rather than vector formulations. Bond Graphs excell in the modeling of systems which include interaction between subsystems of various domains (electrical, pneumatic, mechanical, etc).

the sequencing of submodels) an EPISODE simulator includes a finite state machine.

3.4.1 The Finite State Machine

A finite state machine (FSM) is a system capable of taking on a finite number of states in a dynamically determined (event-driven) sequence of transitions from a particular state to certain others of a set of possible states [4]. A finite state machine is fully specified by its state transition graph, an example of which is shown in Figure 3.2. The finite state model of Figure 3.2 has been drawn to represent the event-driven sequencing of the kinematic constraint conditions in a simplified grand piano action. Only three bodies, H, K, and B, representing the hammer key, and keybed, are considered in this particular FSM. This FSM posesses four states (the four large ovals). The existence of a constraint condition between bodies is noted in Figure 3.2 by a line connecting the body-signifying letters within an oval. The transition paths are denoted with arrows, with a conditional test at the base of each arrow. Starting from a particular state, satisfaction of a conditional belonging to that state will cause the system to take on the state pointed to by the arrow associated with that conditional.

For example, starting in the state in which the key is coupled to the hammer (right oval), either the interaction force between hammer and key f_{KH} will become tensile or the key will hit the keybed (K-B interference) first, depending on how the key is manipulated. From the state in which the hammer is free and the key has not yet hit the keybed (top oval), again: either of the two other states may turn out to be the next state, depending on user interaction at the key.

From this FSM, it is apparent how the constraint conditions may be taken on in various sequences, depending on how other systems (possibly a user) interact with it during run-time. Another advantage of the incorporation of a FSM into a simulator is that it may sometimes be used to reduce the complexity of the submodels themselves. One or more independent generalized coordinates can be dropped from the description of a mechanism if sequencing rules can be deduced from the remaining coordinates. This point will be illustrated with another example taken from musical instrument design: the harpsichord. Figure 3.3 shows the jack of a harpsichord, highlighting each component by name. The operation of the harpsichord jack is quite simple: as the jack is lifted by action of the key, the plectrum meets the string, lifting the string upwards, while bending under the reaction force applied by the string. When the force exceeds a threshold, the string will slip off of the bending plectrum and begin to vibrate. As the jack is once again allowed to lower, the plectrum will not pluck the string, since the spring-loaded tounge and shape of the plectrum will cause the

[4]In the context of the FSM and its use for managing changing kinematic constraints, the word 'state' is taken to mean the operative constraint condition rather than the state vector of generalized coordinates and independent generalized speeds.

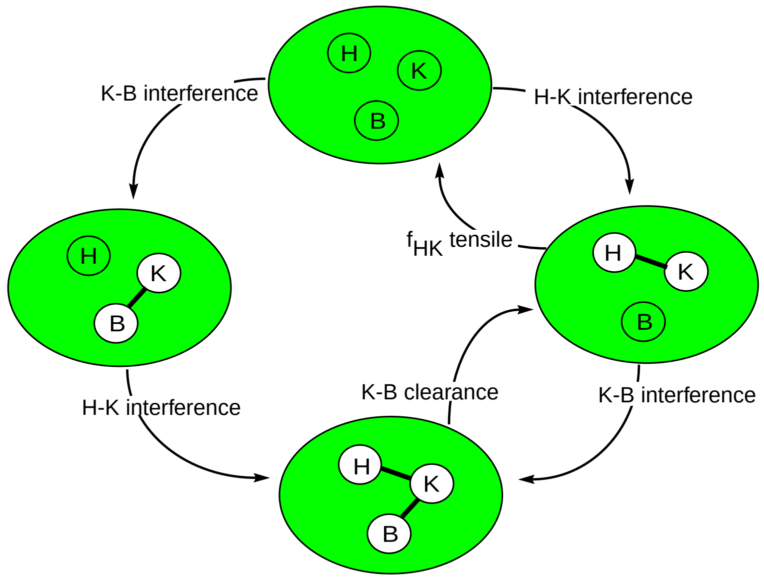


Figure 3.2: *State Transition Graph for a simplified piano action, including a Hammer, Key, and Keybed*

plectrum to easily pivot out of the way upon contacting the string.

To simulate the above mechanism with a dynamical model would entail the modeling of the tongue and the shape of the plectrum, even if the only reaction force one is interested in is the vertical interaction force between key and jack. A simpler approach, incorporating a finite state machine is shown in Figure 3.4. This FSM can be used with a simple static model to create a plucking force on the way up but not on the way down if a pluck has already occured. There is no need to model the tongue.

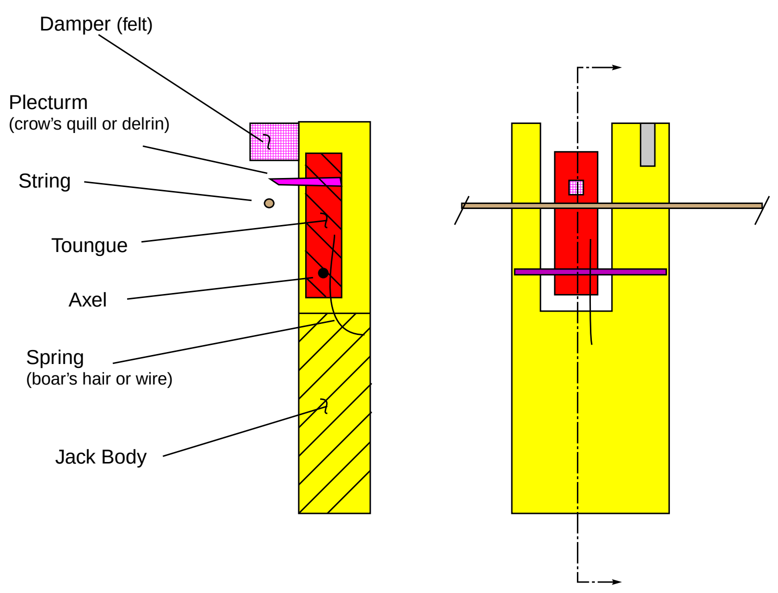


Figure 3.3: *The Harpsichord Jack, shown with plectrum above string (after pluck)*

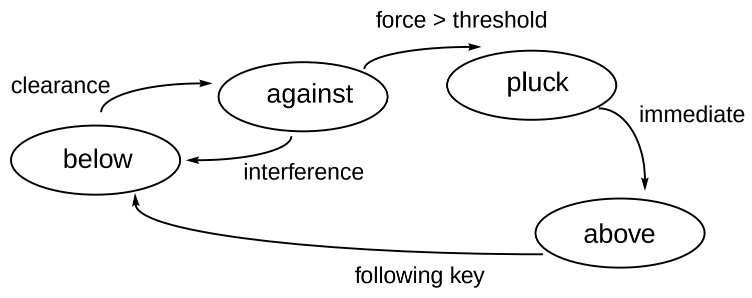


Figure 3.4: *Finite State Machine for the Harpsichord, with state names indicating the position of the plectrum with respect to the string*

3.4.2 EPISODEs

I will now define EPISODEs in detail and outline the construction of a numerical algorithm to solve them. The subscript α (which takes on letters rather than numbers) will be used to enumerate the countable set of submodels $\mathcal{S}$:

$$\mathcal{S} = \{a, b, c...\} \tag{3.7}$$

A complete model is composed of a set of ODEs (submodels), each of which governs the motion in a particular constraint condition.

$$\dot{x}_\alpha = f_\alpha(x_\alpha, u(t)), \quad (\alpha \in \mathcal{S}) \tag{3.8}$$

and a set of readout equations, one for each submodel, which expresses the force output in terms of the state x and input $u(t)$,

$$y_\alpha(t) = r_\alpha(x_\alpha, u(t)), \quad (\alpha \in \mathcal{S}) \tag{3.9}$$

Note that the state x is also subscripted by α since the dimension of x may differ between submodels.

Associated with *each* submodel identified by α is a *set* (indexed by β) of indicator functions which is used to determine the transition times *and* the transition path. The goal of the transition path (a member of $\mathcal{S}$) is denoted by β. Note that the set $\mathcal{T}$ from which β is drawn is a function of α.

$$g_{\alpha\beta}(x_\alpha, u(t)), \quad (\beta \in \mathcal{T}_\alpha), \quad (\mathcal{T}_\alpha \subset \mathcal{S}) \tag{3.10}$$

A *set* of transition functions is used to set up initial conditions for the next submodel from final conditions of the present submodel,

$$h_{\alpha\beta}(x_\alpha) \quad (\beta \in \mathcal{T}_\alpha), \quad (\mathcal{T}_\alpha \subset \mathcal{S}) \tag{3.11}$$

The results of an impulse-momentum solution may be incorporated into the functions h.

The simulator steps forward in time using a numerical ODE solver on the ODE denoted by the present value of α and uses the α readout equation so long as *all* of that ODE's associated indicator functions

$$g_{\alpha\beta}(x_\alpha, u(t)) > 0, \quad (\beta \in \mathcal{T}_\alpha). \tag{3.12}$$

A transition time t_i, $(i = 1, ...)$ signaling the end of the α segment, is found when we detect, for a particular β,

$$g_{\alpha\beta}(x_\alpha, u(t_i)) = 0, \quad (\beta \in \mathcal{T}_\alpha) \tag{3.13}$$

We then switch from the α to the β ODE. The initial conditions for the next ODE are set up as follows:

$$x_\beta(t_i) = x_\alpha(t_i) + h_{\alpha\beta}(x_\alpha(t_i)) \tag{3.14}$$

allowing for addition of state vectors of differing dimension in a straight-forward way.

Of course we cannot find the precise time point at which the event function g_i is identically zero when we are only sampling the indicator function at each integration step. The roots will be crossed over due to the finite step size of the algorithm. If computational time allows, a root finder can be used once a threshold crossing is detected to find t_i to within some prescribed bounds.
Techniques such as these are the subject of Chapters 5 and 6 of this thesis.

3.5 Model Construction

Figure 3.5 shows a profile view of the grand piano action in its rest configuration, highlighting by outline and name each of the elements which will be assumed to be a rigid body in the following analysis. Stick figure representations of each element are also introduced in Figure 3.5. Figure 3.6 shows the same profile view, but highlighting by common name each of the components which will enter our model as a connecting lumped parameter element: springs and dampers. The lumped parameter symbols themselves are also shown in Figure 3.6.

Figure 3.7 shows a schematic representation or stick figure of the piano action, repeated in the same configuration four times. In each subfigure, a different aspect of the model is labeled, but all such aspects are generally applicable. Subfigure a) shows the body and point names, subfigure b) the lumped parameter symbols, subfigure c) the 9 generalized coordinates, and subfigure d) the 27 dimensions. The discussion in each of the following four subsections pertain to a particular subfigure of Figure 3.7.

3.5.1 Generally Applicable Aspects of the Model

Body and Point Names, and Masses

Figure 3.7 a) shows the body names, point names, and any associated masses. The bodies of the action itself are K, W, J, H, and R for the key, whippen, jack, hammer, and repetition lever. E will denote the escapement dolly which is fixed in the newtonian reference frame N. Body M (denoting the manipulandum) is used to drive the model (discussed in further detail below). Because only planar motion shall be considered, a point is sufficient to determine an axis of rotation. Points P_1, P_2, and P_3 are the axes of rotation of bodies K, W, and H, respectively. Point JJ, fixed in W and J, is the axis of rotation for J and point RR, fixed in W, and R, is the axis of rotation for R. Points H^* and K^* are the mass centers of bodies H and K, which carry the masses, M_1,

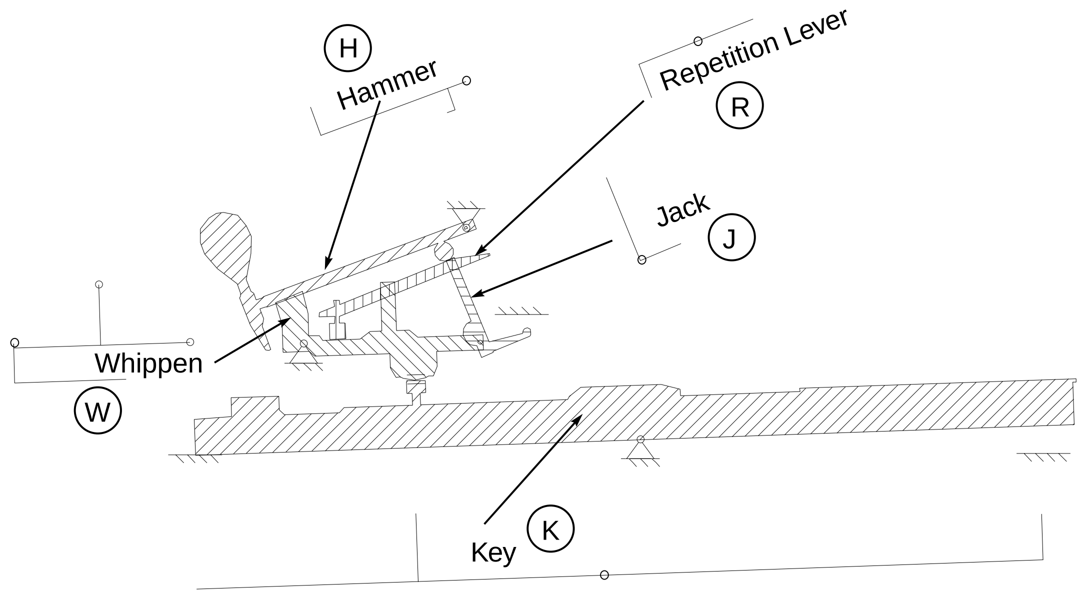


Figure 3.5: *Piano Action Elements: Names, Symbols, and Stick Figures*

and M_2, respectively. Small amounts of mass are assigned to Points RW and JJ to allow equation formulation in 3 degrees of freedom (see Section 3.5.3). Points HJ and JH are the points of contact between H and J, located on H and J, respectively. Similarly, HR and RH are the points of contact (or points of minimum distance, when there is no contact) between H and R, located on H and R, respectively.

Springs and Dampers

Figure 3.7 b) is used to show and define the various springs and dampers which are included in the model. Springs k_2 (keybed back and keybed front), k_6 (repetition lever bed) and k_3 (repetition lever/hammer interface) are special unilateral springs. These unilateral springs are set up with *if* statements in the run-time code. For example, if interference is detected between the key front and the keybed, a spring force proportional to that interference according to coefficient k_2 will act on the key, opposing increased interference. Springs k_2, k_6, and k_7 are typical of the kind of ‘constraints’ used in the coupled force balance formulation. Indeed, the repetition lever R is implemented in the

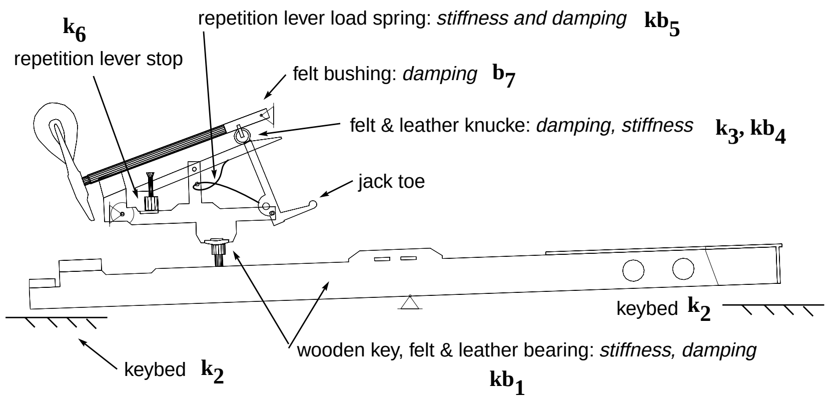

$\mathbf{k_6}$
repetition lever stop
repetition lever load spring: *stiffness and damping* $\mathbf{kb_5}$
felt bushing: *damping* $\mathbf{b_7}$
felt & leather knucke: *damping, stiffness* $\mathbf{k_3, kb_4}$
jack toe
keybed $\mathbf{k_2}$
keybed $\mathbf{k_2}$
wooden key, felt & leather bearing: *stiffness, damping*
$\mathbf{kb_1}$

Figure 3.6: *Piano Action Components*

manner typical of the coupled force balance approach, increasing the number of system degrees of freedom by one. Had the repetition lever been added using a slider between R and H, no extra degree of freedom would have been required. [5]

Likewise, Body M (the manipulandum), pivoted about point P1, has been added to the model (along with the torsional spring k_1 and torsional damper b_1 which couple M and K) in order to conveniently express the force which will be displayed to the user through the haptic display device. Alternatively, body K could have been driven directly by the user, resulting in a zero degrees of freedom system. In that case, the inverse dynamics problem would be solved to determine the reaction forces to this user-determined input motion. The function of the spring damper pair kb_1

[5]Note that the repetition lever has been added to our model using a 'coupled force balance' approach rather than an 'independent coordinate' approach. The ease of implementation using unilateral springs prompted this decision even though the computational efficiency was thereby somewhat degraded. This compromise turned out to be comfortable given our present computing power. In order to implement the repetition lever using the independent coordinate approach, several more states would need to be added to the finite state machine given the various constraint combinations which could occur.

will be further highlighted in Chapter 4.

Rather than using an auxiliary generalized speed to bring the interaction force between J and H into evidence (for use in an indicator function), spring k_4 has been added to the model. The J-H interaction force is conveniently expressed as the extension this spring, q_4 (see Figure 3.7 c)) times k_4. Notice that the addition of spring k_4 has also increased the number of degrees of freedom by one.

Generalized Coordinates

Figure 3.7 c) shows the generalized coordinates q_1 through q_9 which are used to specify the configuration of the 5 bodies. The angle D which locates body M is the specified input. The radian measures of five angles q_1, q_2, q_3, q_5, and q_6 are used to locate each of K, H, R, W, and J with respect to the horizontal. Displacement q_4 is the extension from rest position of the spring k_4. Displacement q_7 locates a frictionless slider $S1$ which connects K to W. Displacement q_8 locates a frictionless slider $S2$ which connects J to the horizontal line E fixed in N. Displacement q_9 locates a frictionless slider $S3$ which connects J to H. The generalized coordinates will be further discussed below in the text pertaining to Figure 3.8.

Dimensions

Figure 3.7 d) is used to highlight all dimensions used in the model.

3.5.2 Comments particular to each submodel or phase

The motion of the piano action will be broken into four phases. Each phase will cover a certain kinematic constraint condition. The phases will be termed ‘acceleration’, ‘letoff’, ‘catch’, and ‘reset’. These phases will also be referred to simply as ‘A’, ‘B’, ‘C’, and ‘D’.

Figure 3.8 shows another four stick figures of the piano action, each one drawn in a configuration typical of one of the motion phases. The phases differentiate themselves from one another by the existence or non-existence of a constraint, and thus in Figure 3.8, the subfigures differentiate themselves from one another by the existence or non-existence of a slider.

All of the sliders $S1$, $S2$, and $S3$ are extant or active only during one phase of the motion of the action: during letoff (as shown in Figure 3.8. Despite the fact that less than 9 coordinates are needed during phases A, B, and D, all 9 coordinates are used, in the same order, for all four submodels. In the case where sliders are missing, generalized coordinates tracking their displacement are not needed. Generalized coordinates are nevertheless used and made to track a ‘would-be’ slider through the use of an ‘artificial’ constraint. In Figure 3.8, the generalized coordinates associated with slider positions (q_7, q_8, and q_9) are shown in all four subfigures. However, when these generalized coordinates are maintained by an artificial constraint (no slider active), they are shown with dotted lines.

For example, q_8 and q_9 are used to locate points BJ and HJ on E and H, which are closest to the head or toe of J, respectively. Using artificial constraints, the passing of the state vector from one submodel to the next may take place without any transition functions h_α. Table 3.1 shows the three phases of motion and the existence of each of the sliders to further clarify the manner in which the kinematic constraints evolve. The manner in which the generalized coordinates q_8 and q_9 are constrained, whether by kinematic loop equation or by artificial constraint is also noted in Table 3.1.

Table 3.1: **Breakdown of constraint equations by type and submodel**

Motion Phase	DOF	Active Sliders	no. slider constraints	no. artificial constraints	artificially constrained
A acceleration	3	S1, S2	4	2	q_8
B letoff	3	S1, S2, S3	6	0	-
C catch	3	S1, S3	4	2	q_9
D return	3	S1	2	4	q_8, q_9

Acceleration

Figure 3.8 a) shows the action in the *acceleration* phase, with slider $S1$ connecting K to W. Note that during the *acceleration* phase, the relative angle between J and W remains constant. The *acceleration* phase ends when the toe of the jack first makes contact with E.

Letoff

Figure 3.8 b) shows the action in a configuration typical of the *letoff* phase. During *letoff*, J and E remain in contact and move with respect to each other on a line, regulated by a slider $S2$. The letoff phase reigns while there is contact between the jack and escapement dolly, and passes to the catch phase either when the interaction forces between the jack and hammer at the knuckle are no longer compressive, or the jack slips out from under the hammer knuckle, which is detected with a threshold on the magnitude of generalized coordinate q_9.

Catch

Figure 3.8 c) shows the action stick figure in a configuration typical of the *catch* phase. This third phase is characterized by free flight of the hammer and further motion of the key, whippen and jack until the jack toe drops below E as detected by a limit on the angle between J and W.

Reset

Figure 3.8 d) shows the action in a configuration typical of *reset*. During reset, only slider $S1$ is at play. Body H continues to settle on R. The reset phase ends when the distance between points HJ and JH drops below a certain threshold, set low to ensure satisfaction of the corresponding constraint which is subsequently enforced –during the follow-on acceleration phase.

3.5.3 Equation formulation

Generalized speeds u_i $(i = 1, ...9)$ are formed as a function of the generalized coordinates simply by setting

$$u_i = \dot{q}_i \quad (i = 1, ...9) \tag{3.15}$$

Six kinematic constraint equations are used to express u_i $(i = 4, ...9)$ in terms of u_r $(j = 1.., 3)$. Expressions are found for the velocities of each of the massive points or points to which spring or interaction forces are applied. The partial velocities for each of these points are found, and are used together with the gravitational forces acting at K^* and H^* and the applied and interaction forces to form the generalized active force F_r $(j = 1.., 3)$. Expressions for the accelerations of K^*, H^*, RR, and JJ are found and used to form the generalized inertia force Fr^* $(j = 1.., 3)$. Finally, the dynamical equations of motion are formulated:

$$F_r + F_r^* = 0, \tag{3.16}$$

[54]. Appendix A contains the *AUTOLEV* input files for each of the submodels.

We are now in a position to link the four submodels during simulation. The generalized coordinates are lined up, so we may pass the final conditions on as initial conditions to whatever submodel comes up next. The indicator functions are presented in Table 3.5.3. Figure 3.9 shows the state transition graph linking the four submodels. During the catch phase, a strike of the hammer on a virtual string is facilitated by a simple *if* statement in the run-time code. A struck string event is demarcated by the sounding of a tone by a synthesizer hooked into our hardware setup. At the time of contact, the sign of u_2, the generalized speed associated with H, is reversed to effect a perfectly elastic collision between hammer and string.

Collisions between bodies which occur at the transitions between phases are assumed perfectly plastic. No impact analysis is performed. The plastic assumption is reasonable in the case of the piano action since, by design, impacting surfaces are covered with felt or leather to avoid impulses.

Table 3.2: **Indicator Functions**

Indicator Function	Expression	Description
g_{AB}	$\mathbf{p}^{JB-BJ} \cdot \mathbf{N}_3 < 0$	vertical distance between jack toe and E
g_{BA}	$q_6 - q_5 - q_{WJ}$ ¡ 0	J-W angle thresholded
g_{BC}	$q_4 < 0$, or $q_9 - threshold < 0$	J-H interaction force tensile or slider slips off edge of knuckle
g_{CD}	$q_6 - q_5 - q_{WJ}$ ¡ 0	J-W angle thresholded
g_{DA}	$\lvert p^{JH-HJ}\rvert <$ tolerance	points JH-HJ within range

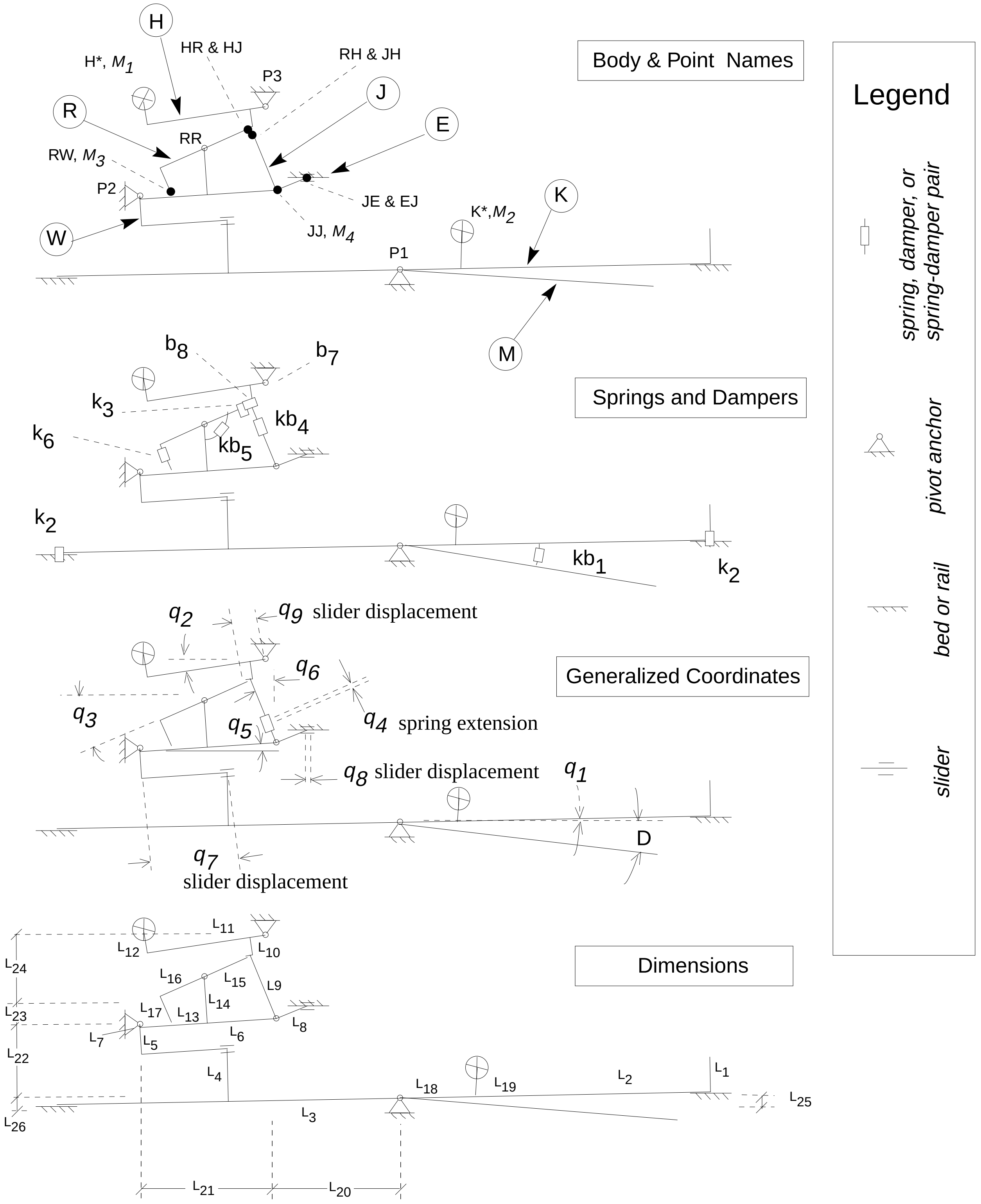


Figure 3.7: *Piano Action Components*

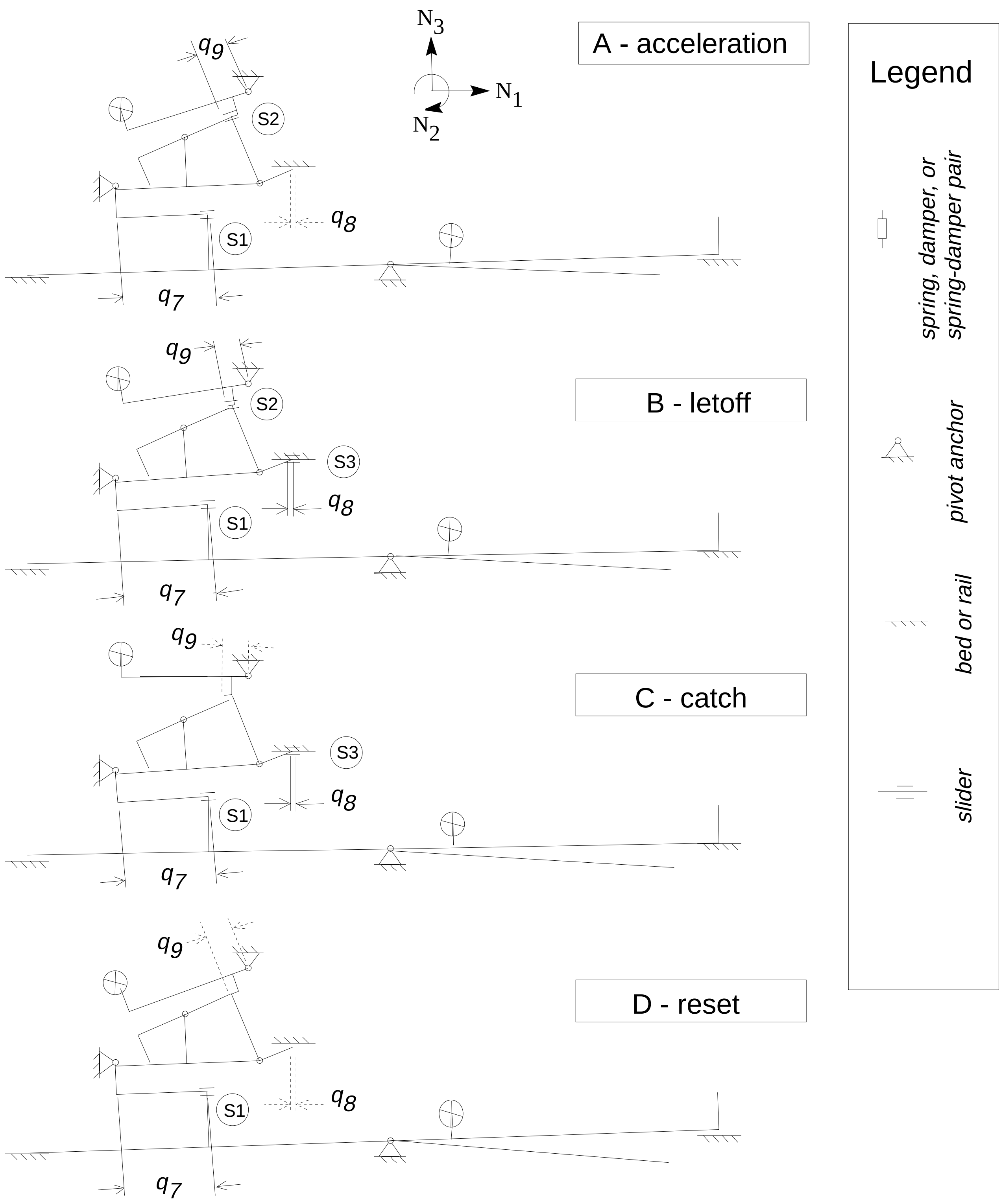


Figure 3.8: *Piano Action Components*

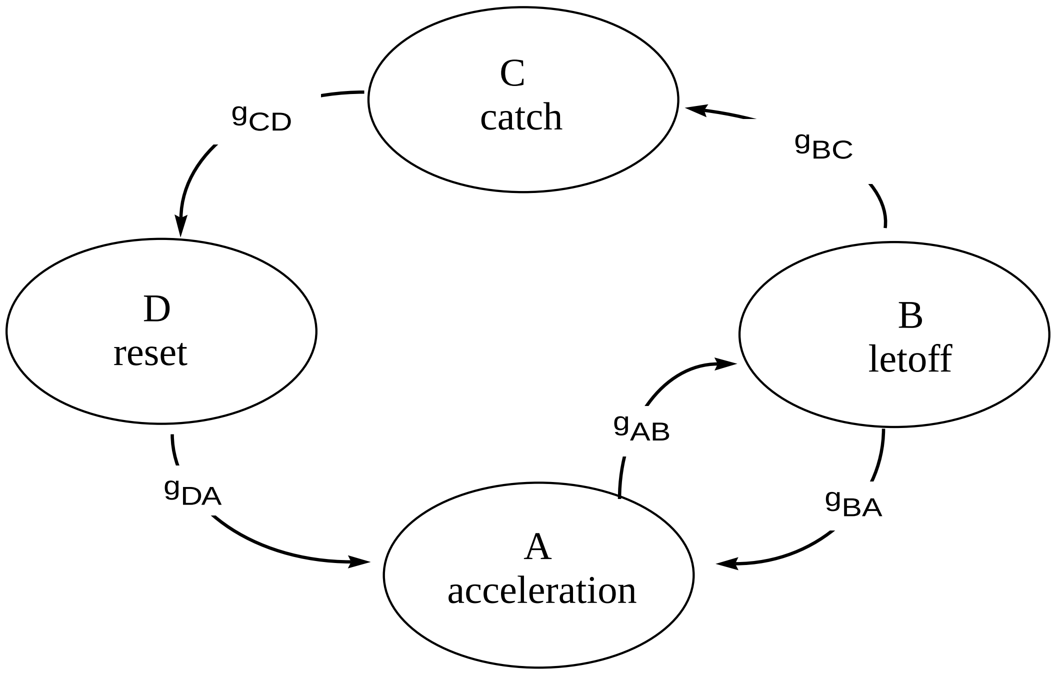


Figure 3.9: *State Transition Graph for the Piano Action*

3.6 Simulator

This section describes the interactive simulator which has been developed for this project. Presently available dynamical system modeling and simulation packages are not designed for real-time simulation, so we built our own simulator from scratch. First, the various components and software tools which are used in the preparation of a new model for simulation are presented. Second, some details of the software architecture are described and finally, some simulation results are presented.

3.6.1 Simulator Components

Figure 3.10 shows the various code components which communicate with the simulator. Basically, construction of a new model for the simulator involves little more than writing the *AUTOLEV* input file, specifying the geometry with AUTOCAD, and invoking various compilers. The simulator then provides for real-time graphical, audio, and haptic interaction with the new model as depicted in Figure 3.11.

Input and Output Files

Paths in Figure 3.10 which terminate on the simulator show the files which are used to add a new model to the simulator's repertoire along with the tools which are used to produce them. *AUTOLEV* 3.0 is used to produce the equations of motion from a command file (model description) as outlined in the previous section. The equations of motion (in the *.EOM file format) are converted into C++ code by a small compiler called *CODEUP* in order to prepare these equations to be compiled with the remaining simulator code [6].

The Model files (*.MDL) contain the geometry to be animated. These files may be generated using AUTOCAD or any other .DXF-format compatible CAD package. Finally, initial conditions and all model parameter values, including dimension, mass, damping, and spring parameter values are loaded in from the (*.DAT) file.

The single stored output from the simulator is a data file shown with an arrow-tail on the simulator block of Figure 3.10. The generalized coordinate, interaction force, indicator function, or virtually any variable trajectory can be stored to disk. Simulation output has been used to place drawings of the bodies into successive frames which have been compiled into Stick-figure animations and full 3-dimensional AUTOCAD animations.

[6] One can imagine future simulator versions which use an interpretive process rather than compilation to incorporate the equations of motion for a new model. For that matter, the task of producing the equations of motion themselves could be taken over by the simulator.

User Interface Devices

The simulator features a host of user interface devices which allow for various kinds of real-time interaction. These are shown linked by double-headed arrow to the simulator in Figure 3.11.

- Sound display is provided through a MIDI-driven synthesizer.
- A graphical user interface facilitates the modification during run-time of virtually all parameter values, even control values such as the time step and display gains. Figure 3.12 shows an example simulator control dialog box. Note that space-efficient access to all parameter values is provided by pull-down buttons.
- An electro-mechanical apparatus employing motors coupled to keys, the design of which will be featured in Chapter 4, provides for haptic interface.
- A scope view provides for real-time graphing of various generalized coordinates or other variables.
- A model view module provides for real-time animation of the geometry loaded from the Model file (*.MDL). We have found real-time visual display to be invaluable for debugging simulated behavior. Synchronous haptic and visual display can provide many clues when something is wrong.

3.6.2 Some Details of the Software Design

The simulator was developed for MicroSoft-Windows using the Borland C++ compiler to run on a Pentium 90 MHz PC. The equations of motion and expressions for the interaction forces are integrated numerically using a Fourth-Order Runge-Kutta algorithm. Care must be taken to ensure that the initial conditions do indeed satisfy the applicable constraint equations. This is accomplished by solving the set of non-linear non-differential constraint equations numerically by the method presented in [54], page 222.

A dialog box chosen from the main menu from several available virtual objects becomes the main simulation controller. The simulation dialog box acquires its functionality by composition rather than by inheritance (See Figure 3.13.) Upon selection of say, the virtual piano action, each of the pertinent submodel objects (differential equation to be integrated) is constructed and composed into a finite state machine. Because the members of the graphical objects, models, and indicator functions are all written as virtual member functions, the simulator object can take advantage of the run-time polymorphism features of C++. Thus, the simulator engine itself is completely generic code, able to operate on any of the models. Once created, a simulator dialog box will be called upon at each iteration of the event loop to execute the following: poll the sensors for current readings, use the current submodel to integrate ahead one time-step, check the indicator function and change submodels if necessary, send a force value per the readout equation to the D/A converters, and finally, perform graphic and sound display functions.

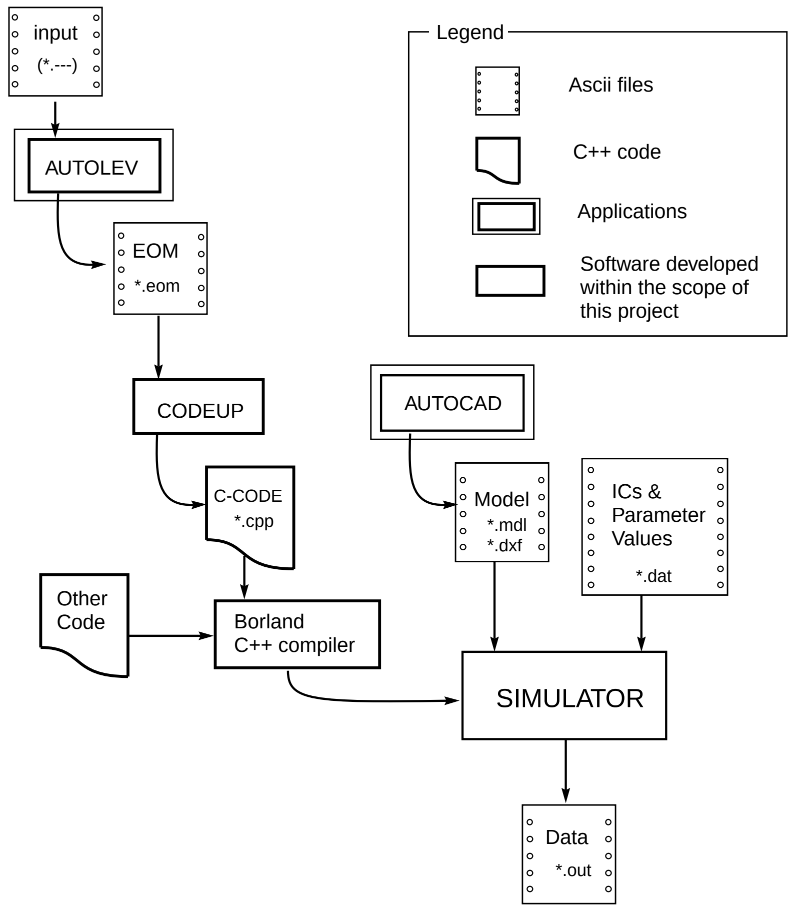


Figure 3.10: *Input and Output Files for the Simulator, and their Producers*

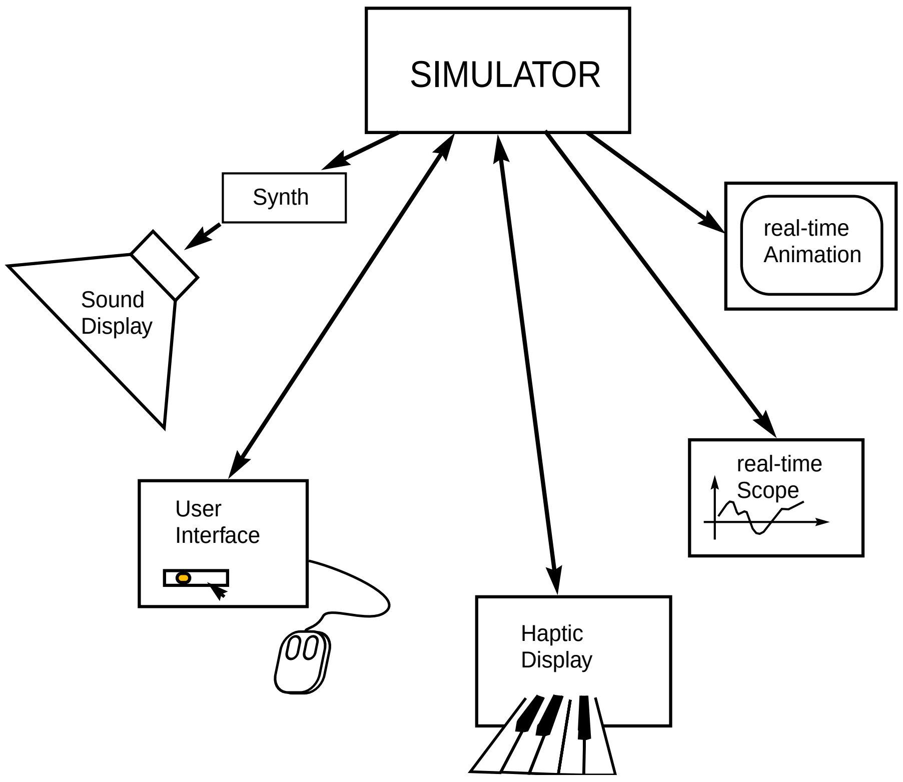
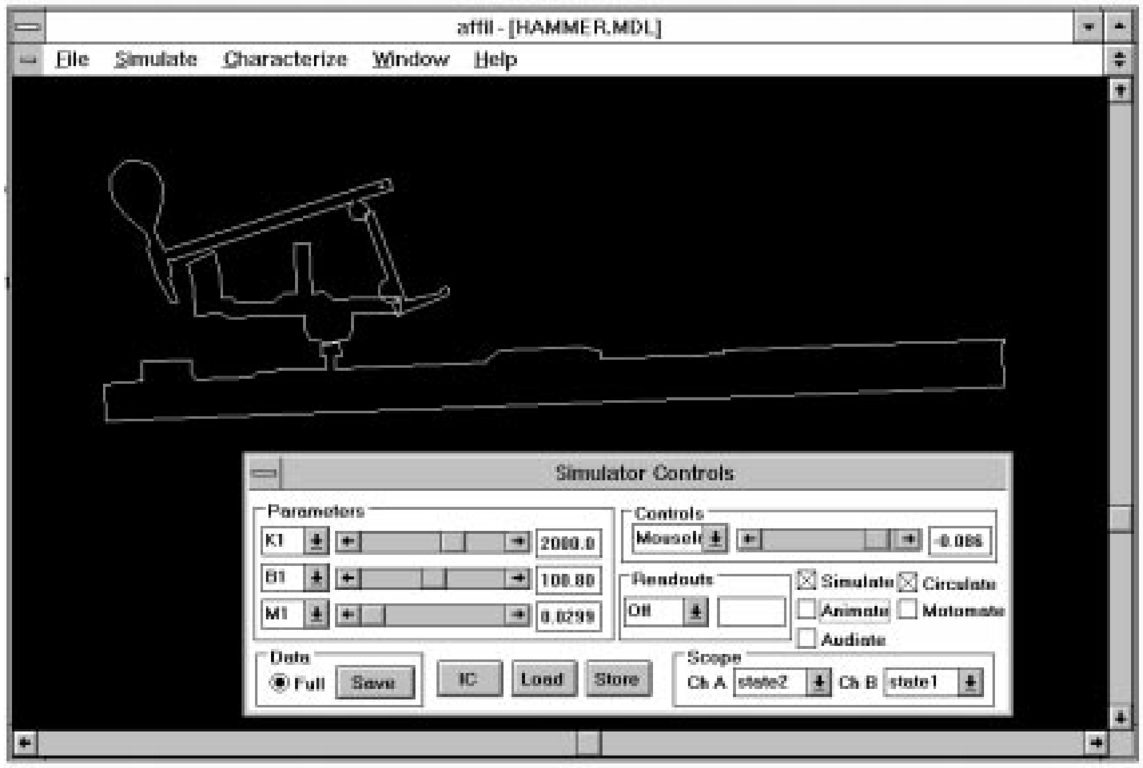


Figure 3.11: *Interface Devices for the Simulator*



K1 2000.0
B1 100.00
M1 0.0299
Mousel -0.086
Off
[✓ Simulate] [✓ Circulate] Animate Motomate Audiate
[✓ Full]
Ch A state2 Ch B state1

Figure 3.12: *Interface Devices for the Simulator*

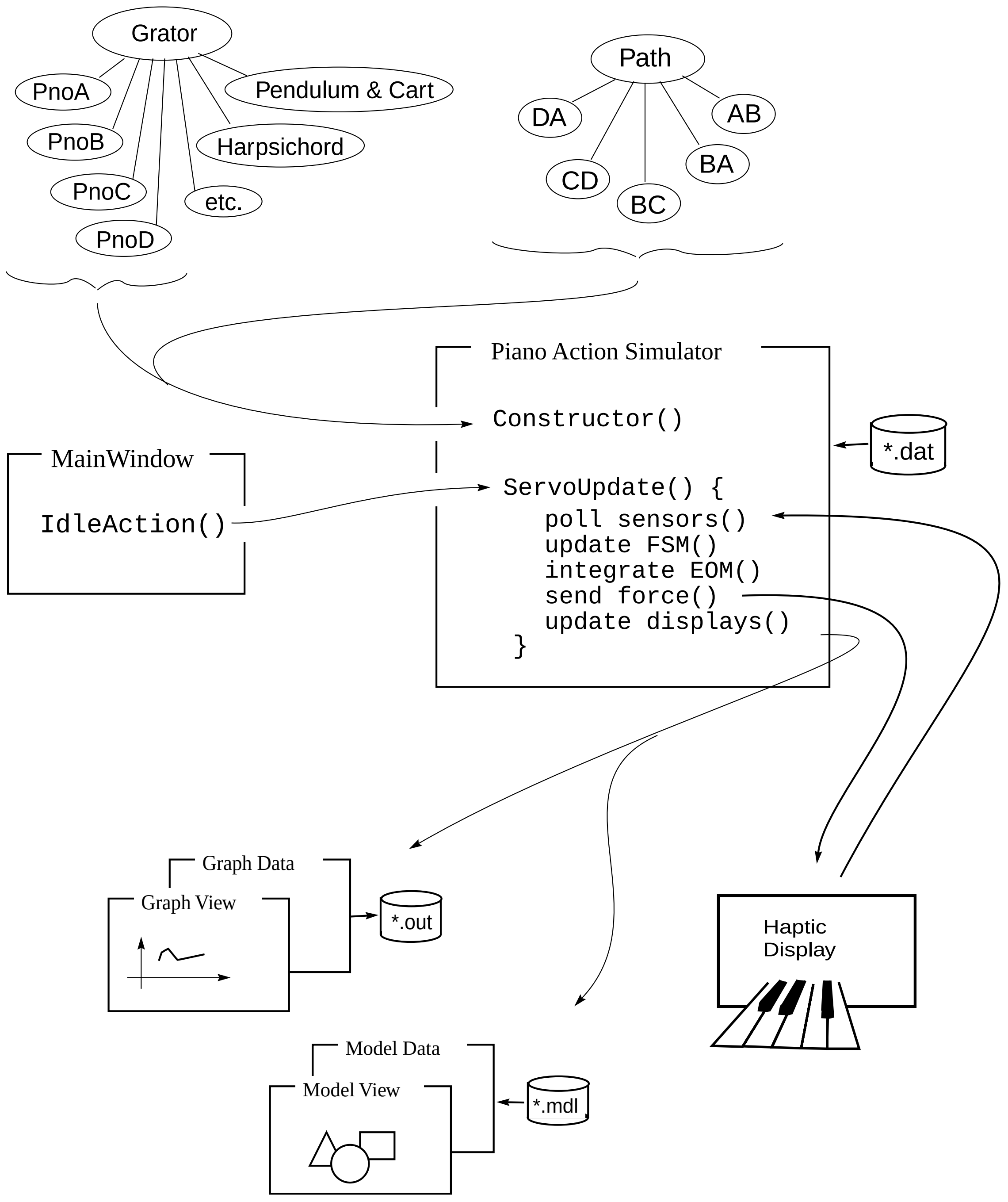


Figure 3.13: *Software Architecture for the Simulator*

3.6.3 Sample simulation output

In this section, the results of a simulation of the grand piano action model will be presented. A pre-specified motion input of the manipulandum will be applied, and the response studied and compared to experimental results.

Values for the parameters L_1 through L_{27} which were used for the simulation are given in Table 3.3. Figure 3.14 shows the method which was used to determine values for each of the dimensions, and to determine the initial simulation values for the generalized coordinates. A plan-view video image of an isolated one-key piano action model (of the kind often displayed in piano showrooms) was digitized and imported into a *DRAW* program on the NeXT computer. Bold lines corresponding to the stick Figure were drawn over the image and a query tool available from the *DRAW* program was used to determine line dimensions in pixels and angles. These dimensions were then scaled to meters using certain known (directly measured) values.

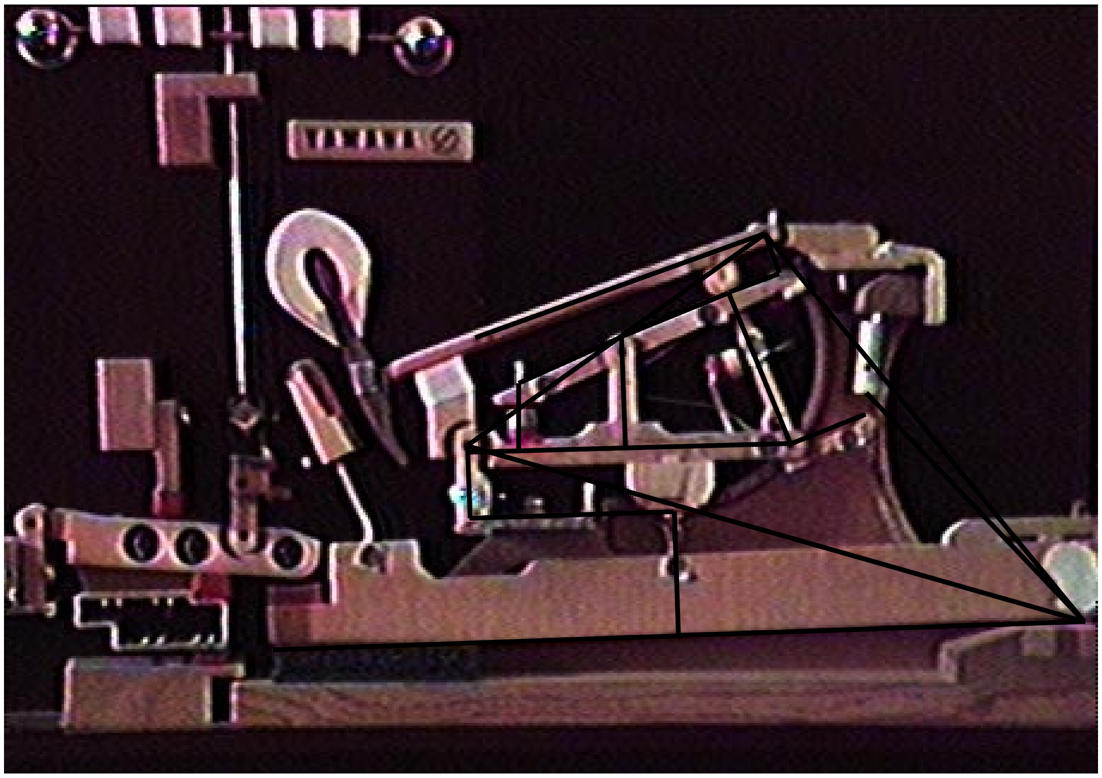


Figure 3.14: *Method for extracting certain dimensions and initial configuration angles from the actual piano action*

Table 3.4 gives the values used for the lumped parameters. These values were chosen for the

Table 3.3: **Parameter Values**

Dimension	Value [m]
L_1	0.0262
L_2	0.2286
L_3	0.1260
L_4	0.0386
L_5	0.0223
L_6	0.1001
L_7	0.0025
L_8	0.0250
L_9	0.0508
L_{10}	0.0129
L_{11}	0.0879
L_{12}	0.0170
L_{13}	0.0493
L_{14}	0.0366
L_{15}	0.0354
L_{16}	0.0351
L_{17}	0.0208
L_{18}	0.0556
L_{19}	0.0117
L_{20}	0.0988
L_{21}	0.0922
L_{22}	0.0544
L_{23}	0.0135
L_{24}	0.0521
L_{25}	0.0066
L_{26}	0.0064

most part by trial by error, although some guidelines were followed. Note that the stiffnesses used for spring-damper couplers (k_3 and k_6) are large. The 'force sensor' used for the *J*-*H* interaction force (k_4) was chosen quite high. The masses of the hammer and key are consistent with physically measured values. Damping coefficients were chosen generally by trial and error.

Table /refTab:InitialConditions shows the initial conditions used for simulation. These were derived as described in the text pertaining to Figure 3.14.

Figure 3.15 and 3.16 show the results of a simulation using the piano action model developed in the previous section. Figure 3.15 shows the trajectories of generalized coordinates q_1, q_2, q_3 and q_6 (the angles which the key, hammer, repetition lever, and jack make with the horizontal, respectively). Submodel simulation phases A, B, and C are also noted at the bottom of Figure 3.15. Also shown in Figure 3.15 is the driving input D (the angle which the manipulandum makes with the horizontal)

Table 3.4: **Parameter Values**

Stiffness Symbol	Value [N/m]	Damping Symbol	Value [N/m/s]	Mass Symbodl	Value [kg]
k_1	200.0	b_1	0.82	m_1	0.014
k_2	80.0			m_2	0.120
k_3	600.0			m_3	0.01
k_4	2000.0	b_4	0.2	m_4	0.05
k_5	0.35	b_5	0.06		
k_6	600.0				
		b_7	0.001		
		b_8	8.0		

Table 3.5: **Initial Conditions**

Generalized Coordinate	Value	Units
q_1	-0.03473	radians
q_2	-0.35622	radians
q_3	-0.38572	radians
q_4	0.0	meters
q_5	-0.03684	radians
q_6	-0.389557	radians
q_7	0.06223	meters
q_8	-0.0000254	meters
q_9 0.01649	meters	

used for this particular simulation run . Generalized coordinates D and q_1 are shown scaled by a factor of 4 to make their features apparent in Figure 3.15.

At time $t = 0.1$ seconds, the input D (manipulandum angle) begins to ramp up from -0.0335 radians. (During interactive simulation, D would be driven by the user, sensed by an encoder on the manipulandum). The ramp continues to $t = 0.18$ seconds and stops at -0.0055 radians.

Phase A acceleration

When the model simulation is initiated, a small amplitude transient is observed. This transient is due to the sudden application of gravity. As D ramps up, q_1 follows. Generalized coordinate q_2 (hammer) rises at a rate which about 5 times greater than q_1, consistent with the 5 times mechanical advantage which the key has over the hammer. Generalized coordinates q_3 and q_6 both decrease during phase A. Acceleration ends at $t = 0.1334$ seconds.

Phase B letoff

During letoff, q_1 continues to rise and q_2 rises at 5 times that rate. But now, q_3 and q_6 proceed in opposite directions. Generalized coordinate q_6 takes a sudden turn in direction at the transition from phase A to B. This point in time corresponds to the jack toe having met the escapement dolly. During phase B, by action of the constraint which prevents the jack toe from penetrating the dolly, the jack head begins to pivot out from under the hammer knuckle.

The end of phase B is signaled when the jack head has slid off the end of the hammer knuckle which occurs at $t = 0.181$ seconds. This event corresponds closely in time with the end of the ramp input on D, a chance occurance whcih was an effect of the particular choice of driving input for this model.

Phase C catch

At the beginning of phase C, when the hammer is first decoupled, the assembly of all elements but the hammer makes a somewhat abrupt move, quickly settling to its final configuration. This effect has to do with the sudden release of the action of damper b_8, which was impeding the sliding motion of Slider S_2 (between jack and hammer) during letoff.

At approximately $t = 0.2$ seconds, the hammer strikes the string (which is positioned at $q_2 = 0$ degrees) as seen by the reverse in direction of trace q_2. Thereafter, the hammer settles on the repetition lever. Low frequency oscillations are observed in q_2 and q_3. Very small amplitude oscillations are also evident in q_6 and q_1.

Indicator Functions

Figure 3.16 shows the traces of two indicator functions, g_{AB} and g_{BC}. Function g_{AB}, as the reader will recall from Table 3.5.3, is simply the vertical distance between the jack toe and the escapement dolly. When g_{AB} drops to zero, (which occurs at $t = 0.1445$ seconds) the transition from phase A to B is signalled. Function g_{AB} remains at zero (to satisfy the constraint associated with slider S2) during phases B and C.

Function g_{BC} is the displacement of slider S_3 (link between jack head and hammer knuckle) from the edge of the knuckle. When g_{BC} drops to zero, (at $t = 0.181$ seconds) the jack head has slipped out from under the knuckle.

Figure 3.17 shows the simulaed interaction force trajectories. The extension of the parallel spring-damper pair k_1-b_1 is simply $(q_7 - q_1) * k_1$. This is the force which would be displayed to the user through a hapter interface in an interactive simulation. The letoff phase corresponds to a period of increased force. This is the 'letoff resistance'. The trace of $q_4 * k_4$ is the interaction force

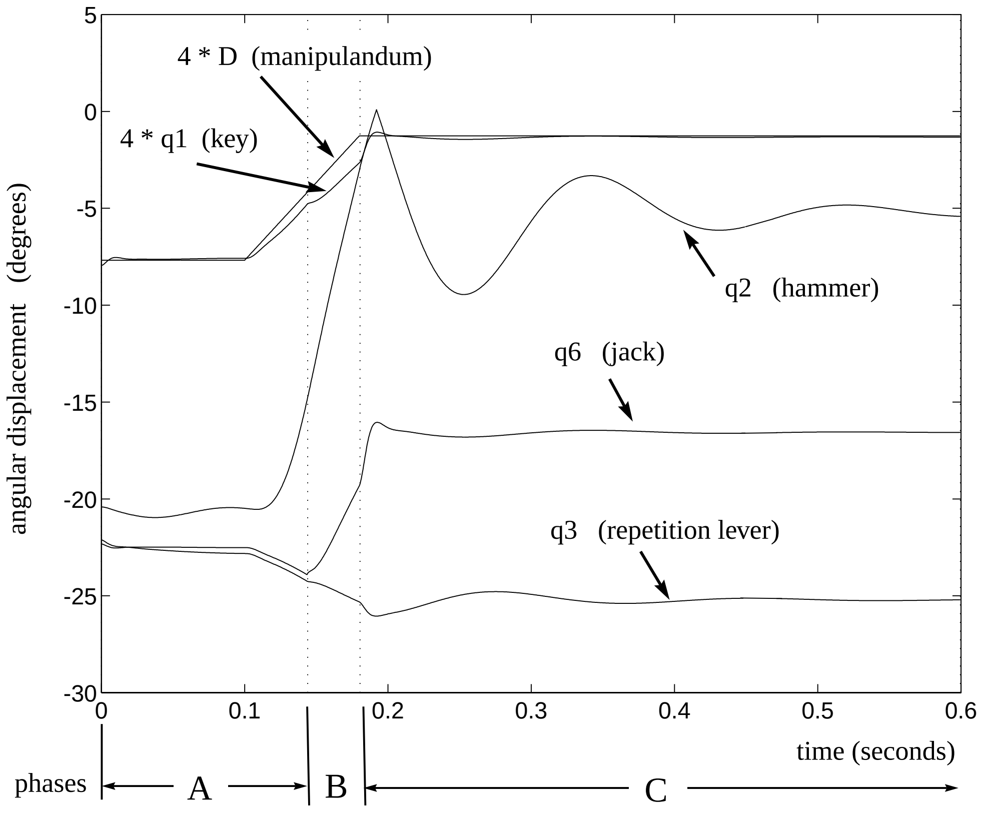


Figure 3.15: *Simulated Generalized Coordinate Trajectories*

between jack and hammer, along the axis of the jack. This force remained compressive throughout the entire simulation run, which is a function of the driving input. Had this trace wandered below zero, a release of the hammer would have occured by the 'other' $g_B C$ indicator function (see Table /refTab:IndicatorFunctions), $q_4 < 0$. Instead, release of the hammer occured by the jack slipping out from under the knuckle (q_9 threshold).

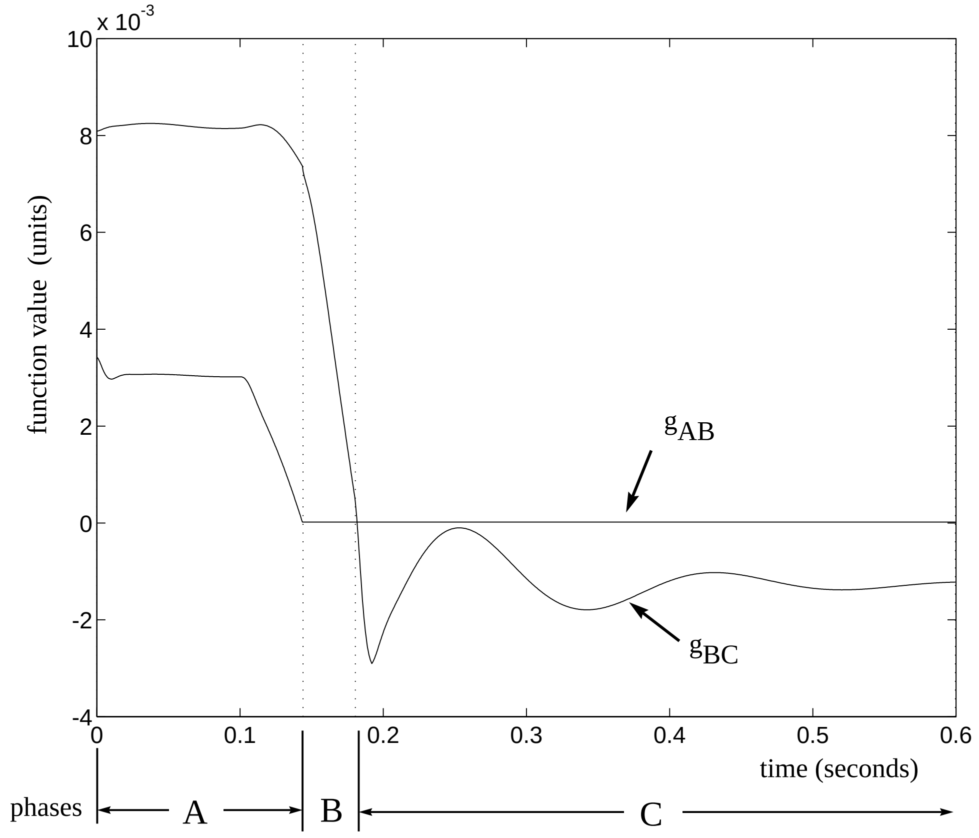


Figure 3.16: *Simulated Indicator Function Trajectories*

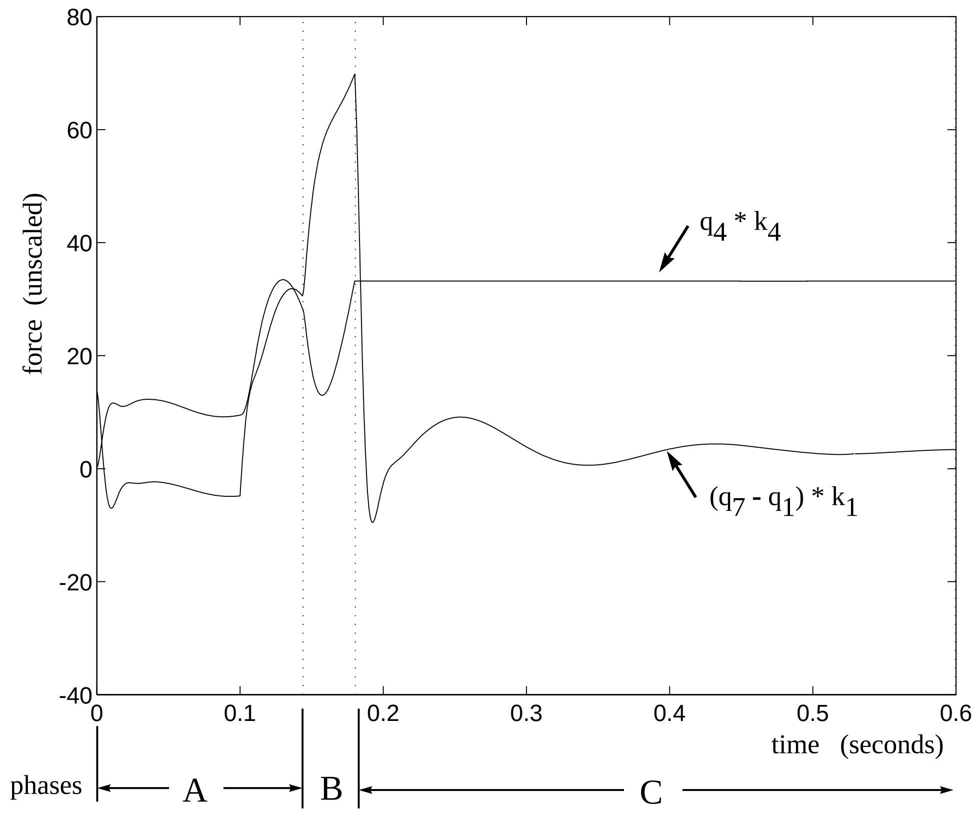


Figure 3.17: *Simulated Interaction Force Trajectories*

3.7 Experiment and Simulation Comparison

An isolated one-key grand piano action (see Figure 3.14) was used in an experiment to produce data against which the simulation output could be checked. The piano action was set into motion by releasing a weight from rest just above the key. The resulting motion of each action body was recorded using a high-speed video camera at 1000 frames per second. Retro-reflective patches were attached to ten locations on the piano action in order to facilitate vision recognition by computer. Illumination by bright lights and sensitivity adjustments on the camera produced an image for recording of 10 bright moving light patches on a dark background. Digitization and light patch centroid location determination from about 700 frames for each sequence was performed by Jim Walton of 4-D Video, Sebastapol, California. The digitized motions were used to deduce corresponding generalized coordinate trajectories with inverse trigonometric transformations. These experimentally determined generalized coordinate trajectories are shown in Figure 3.18.

As the key rises, the hammer rises at about 5 times that rate during the initial period of motion. Initially (presumably during acceleration) the jack and repetition lever (q_6 and q_3) both decrease. Then (presumably during letoff) q_6 and q_3 move in opposite directions. Finally, the hammer strikes the string (at $t = 0.2$ seconds into the recording) and then settles on the repetition lever, as seen by the oscillations in q_3 and q_2 after the string strike.

3.7.1 Discussion

Inspection of Figures 3.15 and 3.18 show strong similarity between simulated motion and experimental motion. But, as stated earlier, parameter values for simulation were chosen by trial and error to make the simulation results similar to the experimental data. It cannot, at this point, be claimed that the model will necessarily produce behavior indicative of its referent, when physically meaningful parameter values are chosen, only that similar motion *can* be produced by careful parameter selection.

Parameter selection can be an arduous process for a model with such complex behavior, but precisely because this model so closely follows the physical piano action in form, all adjustments may be expected to have intuitive (appropriate) effects.

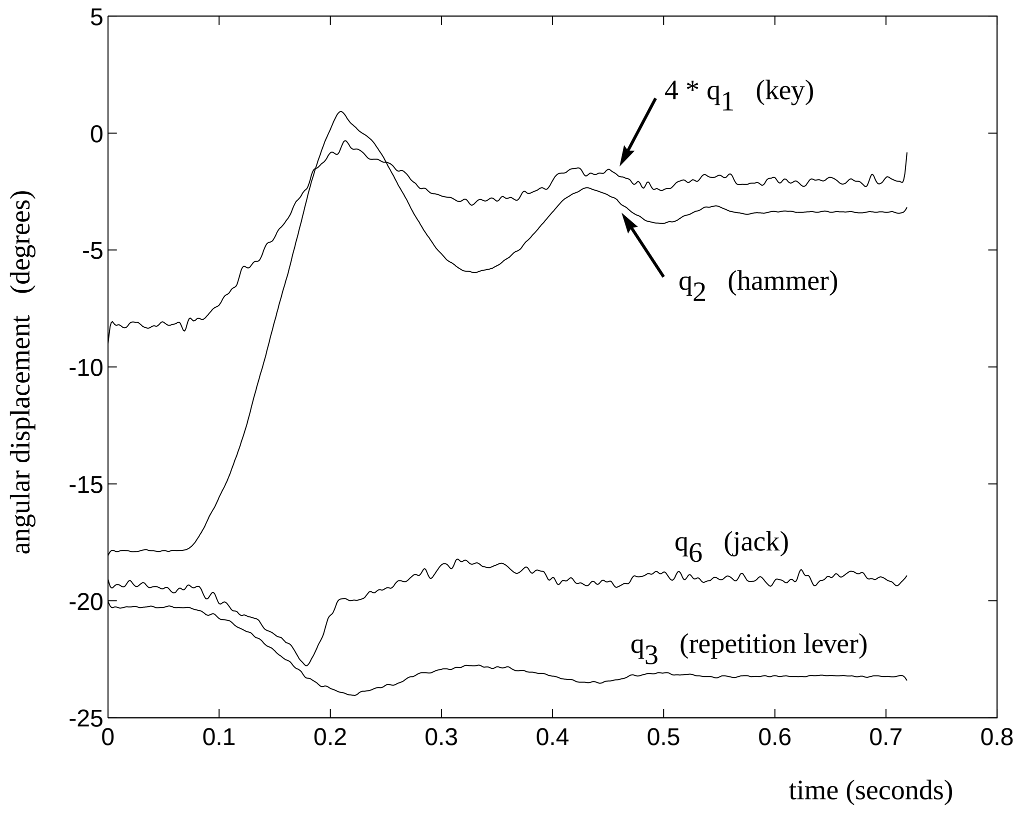


Figure 3.18: *Experimentally determined generalized coordinate trajectories*

3.8 Summary

An overview of the various model formulations for multibody dynamical systems revealed a tradeoff between ease of handling changing kinematic constraints and computational efficiency. Simulation through constraint changes of the most numerically efficient model form, the independent coordinates formulation, was identified as an area which has received little attention in the literature.

A modeling and simulation algorithm based on a model in independent coordinates has been presented which accommodates dynamical systems with changing kinematic constraints. Systems for which the various constraint conditions may be pre-determined but the ordering of constraints is left as a function of run-time conditions are handled. Submodels are constructed for the system in each of its constraint conditions in the independent coordinate formulation, providing for maximally efficient numerical simulation. An ODE solver and a set of submodel management routines in the form of a finite state machine are used for simulation, which may be run interactively, with multiple user input and output devices.

Various escapement mechanisms fit into this class of systems. A five-body model of the piano action was presented and used as an illustrative example of the simulator. This model is actually an hybrid of the coupled force balance and independent coordinate model formulations. The repetition lever uses coupling spring-damper pairs. The model does indeed exhibit behavior suggestive of its referent, the grand piano action, as shown through comparison of simulation and experimental data when parameters are chosen carefully.