AIAA JOURNAL
Vol. 36, No. 2, February 1998

Physical Programming for Computational Control

Achille Messac* and Bruce H. Wilson†
Northeastern University, Boston, Massachusetts 02115

A new approach to design optimization, physical programming, has recently been developed. Physical programming allows the designer to express design preferences explicitly for each design metric of interest, e.g., settling time or control effort, in a flexible and physically meaningful manner. This flexibility allows the designer to introduce a quantitative degree of desirability into the problem formulation by using the categories highly desirable, desirable, tolerable, undesirable, highly undesirable, and unacceptable and by defining numerical ranges that fit in these categories for each design metric. In this study we demonstrate the use of physical programming for controller synthesis by solving a benchmark control problem. We implement a physical-programming-based problem solution, where a preference function is developed for each design metric. These metrics include nominal and worst-case settling time, nominal and worst-case control effort, worst-case noise amplification, and robust stability. We design controllers of three different orders and compare these against published controller solutions of the same order for the benchmark problem. Physical programming control synthesis methods are used to design high-performing and robust controllers with remarkable ease, and the controllers compare quite favorably with published solutions.

I. Introduction

MOST control synthesis methods are based on an underlying theory. Linear quadratic regulator (LQR), linear quadratic Gaussian (LQG), H_∞, and quantitative feedback theory are but a few such examples; each harbors its own strengths and weaknesses. In a different vein, computational control can be regarded as an approach to control design that does not rely explicitly or exclusively on a single underlying control synthesis theory. Instead, computational control uses nonlinear programming as the basis for its search for the optimal solution. The particular computational control approach chosen may use any combination of appropriate control theories. For example, the LQR approach can be chosen as the underlying theoretical basis, while the state and control weighting matrices are obtained by minimizing the appropriate aggregate objective function, subject to relevant constraints. Computational control offers key advantages. The designer can explicitly minimize or maximize the quantities of interest, e.g., settling time and control effort, rather than hope for a desired behavior of those quantities through indirect metrics such as phase margin and closed-loop bandwidth.

When the control problem is well posed and the designer is an expert in the area of nonlinear programming, computational control might be as competitive as any other method. Ironically, the key shortcoming of computational control is its very dependence on nonlinear programming. Too often, the successful application of nonlinear programming depends on the level of expertise of the user, not in the discipline of interest, but in the area of nonlinear programming. Because the expertise of most control designers is in control rather than in nonlinear programming, it is understandable that computational control has not enjoyed great popularity, despite its truly significant capabilities. By removing significant impediments in the application of nonlinear programming, this paper breaks new ground to redress this situation.

A new approach to computational design optimization entitled physical programming has recently been developed by Messac.[1] By employing a flexible and more natural problem formulation framework, physical programming is intended to reduce design time, to reduce computational effort, and to greatly enhance the designer's ability to obtain an optimal design. Physical programming possesses the desirable feature that a designer does not need to specify optimization weights in the problem formulation phase. Rather, for each design metric, the designer specifies numerical values for ranges of differing degrees of desirability.

Consider typical design metrics such as settling time t_s and maximum controller effort $u_{\max}$. Design specifications may require $t_s \leq 20$ and $u_{\max} \leq 50$, and a designer may also wish to minimize each metric, without violating the hard specification. The designer creates an aggregate objective function of the form

$$J = \alpha t_s + \beta u_{\max}$$

and manipulates the weights α and β until the minimized aggregate objective function leads to acceptable values for t_s and $u_{\max}$. Designers may introduce considerable bias or subjectivity into the problem. They may accept the values of 19.9 for t_s and 30.2 for $u_{\max}$ but may much prefer 15.0 for t_s and 34.0 for $u_{\max}$. However, they cannot explicitly articulate this wish using the traditional aggregate objective function shown earlier. Relying on weights, rather than explicitly articulating preferences, is a key limitation of conventional optimization.

An alternative to the preceding scenario is to monitor the optimization process and to change the weights during the solution process periodically, as opposed to performing repeated full optimization runs. Because of its associated difficulties, this approach offers little appeal to an optimization expert. It is even less appealing to a control engineer whose primary objective is to obtain a design that works well, rather than to confront the intricacies of mathematical programming. Moreover, when the designer must deal with a large number of conflicting objectives, the task of tweaking the weights for the different design metrics becomes prohibitive.

By allowing the designer to express preferences flexibly and concisely, physical programming addresses the aforementioned limitation of traditional optimization. Simultaneously, physical programming obviates the need to adjust weights. We first describe how a designer might express a realistic design preference. The preference ranges for the settling time (in seconds) of an impulse response could be stated as follows: highly desirable, <12; desirable, 12–14; tolerable, 14–16; undesirable, 16–17; highly undesirable, 17–20; and unacceptable, >20. Note that this description encompasses both hard specifications and soft objectives. The latter are expressed in terms of ranges of differing degrees of desirability.

In addition to eliminating the need to specify and to adjust weights, physical programming is also well suited to address the inherent multiobjective nature of design problems, where multiple conflicting objectives govern the search for the best solution.

Physical programming provides a more deterministic approach to obtaining a solution that satisfies the typically complex texture

Received Jan. 17, 1997; revision received Aug. 10, 1997; accepted for publication Sept. 24, 1997.

*Associate Professor, Department of Mechanical Engineering, Multidisciplinary Design Laboratory. E-mail: messac@coe.neu.edu. Associate Fellow AIAA.

†Assistant Professor, Department of Mechanical Engineering, Multidisciplinary Design Laboratory. E-mail: wilson@neu.edu.

of a designer's preferences. A detailed development of the physical programming approach is provided in Ref. 1. Reference 2 illustrates an application of physical programming in the case of the high-speed civil transport plane. A synopsis of the physical programming implementation procedure is provided in the sequel, with a bias toward its intended control application in this paper.

The continual adjustment of weights encountered during traditional optimization also occurs during control design. For example, during an LQR controller design, it is necessary to choose weight matrices for the state and control variables. Because of the typical coupling among these variables, the process of uncovering the weights that adequately satisfy a realistic set of conflicting objectives is often a nontrivial task. To date, no definitive approach to address this problem exists. Another example is the integrated control and structural design of dynamical systems.[3–6] This problem is highly intensive computationally: A single optimization run on a workstation could last from several hours to several days. The idea of iterating on weights in the context of that work is at best impractical and exposes a serious impediment to routine application of multidisciplinary optimization technology in industrial settings.

Physical programming has not been applied to the design of control systems. This paper is the first to explore the effectiveness of physical programming to computational control design. As discussed earlier, the idea of using mathematical programming as an approach to solving control problems is not new. Early work in that area was presented by Tabak and Kuo[7] in a 1971 book that summarizes much of the work of the previous nine years. A more recent treatment of the subject is presented by Bulirsch and Kraft,[8] where a comprehensive picture of the state of the art can be found, both in terms of software and in terms of theory. The work of this paper is a new development, as it is the first application of physical programming to controller synthesis. Previous work in this area used such methods as generic nonlinear programming or goal programming.[8]

Physical programming's ability to cope with multiple, conflicting objectives makes it well suited to address the two-mass benchmark problem introduced by Wie and Bernstein.[9] The problem involves the control of a mass–spring–mass system with uncertain properties, subject to various disturbance inputs. The problem presents generic control issues and is clearly formulated. A number of researchers have explored the application of various methods to this problem and proposed various solutions, which offers a convenient means to compare results obtained from physical programming. Wie and Bernstein[9] cite approximately 13 solutions. Stengel and Marrison[10] cite some six solutions. Reference 11 contains 12 articles on the benchmark problem. The conflicting physical objectives that the problem poses, e.g., $t_s \leq 15$, while $u_{\max}$ is reasonable, place it in the category of problems physical programming solves.

This paper is organized as follows. Section II presents a brief outline of the procedure for applying physical programming. Section III defines the two-mass benchmark problem, together with the key governing equations. The method of investigation is developed in Sec. IV. The controller design results and discussion thereof are the subjects of Secs. V–VII. Concluding remarks are given in Sec. VIII.

II. Physical Programming Application Procedure

This section describes the procedure for applying physical programming, assuming an operational software implementation thereof is available. Alternatively, the theoretical development of physical programming developed in Ref. 1 can be coded by the user. The physical programming implementation that is used for this paper is embodied in the software package entitled PhysPro,[12] which is Matlab[13] based and is at a more advanced stage than that which is presented in Ref. 1. To apply physical programming to any problem, several well-defined steps must be performed. We begin with some requisite preliminary discussions and definitions.

A. Definitions

1. Decision Variables

To perform the design, it is assumed that the designer has the ability to alter a set of system features represented by the variables x_i, which are components of the vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_p)$. The vector $\boldsymbol{x}$ is variously referred to as the design parameter, the design variable, or the decision variable vector. The choice of the design variable set plays an important role in the design process. The omission in $\boldsymbol{x}$ of critical design features may severely jeopardize the ultimate outcome. Conversely, the inclusion in $\boldsymbol{x}$ of design features that do not appreciably impact the design performance will unduly burden the computational process. The experience and intuition of the designer in the technical design area under consideration are of high value. In the control design work of this paper, the design variables are the coefficients of the polynomials of the controller transfer functions.

2. Design Metrics and Preference Functions

The problem formulation also involves identifying characteristics of the system or design that allow the designer to judge the effectiveness of the outcome. These characteristics, or design metrics, are denoted by g_i, which are components of the vector $\boldsymbol{g} = (g_1, g_2, \ldots, g_q)$. The elements g_i represent system behavior. Design metrics may be quantities that the designer wishes to minimize; maximize; take on a certain value (goal); fall in a particular range; or be less than, greater than, or equal to particular values. The designer defines the preference functions by specifying these ranges and values. Each design metric may become part of an aggregate objective function that will be minimized, or may be used in equality or inequality constraints, or may be both. Any subset of $\boldsymbol{x}$ may be a subset of $\boldsymbol{g}$, making it possible to express preferences with regard to the design parameters as well.

3. Classification of Metrics and Class Functions

Within the physical programming procedure, the designer expresses preferences with respect to each metric using four different classes. Figure 1 depicts the qualitative meaning of each class. The value of the metric under consideration, g_i, is on the horizontal axis, and the function that will be minimized for that metric, $\bar{g}_i$, hereby called the class function, is on the vertical axis. Each class comprises two cases, hard and soft, referring to the sharpness of the preference. All soft class functions will become constituent components of the aggregate objective function.

The desired behavior of a generic metric is described by one of eight subclasses, four soft and four hard. These classes are illustrated in Fig. 1 and are characterized by the following.

Soft:

Class-1S = smaller-is-better (minimization)
Class-2S = larger-is-better (maximization)
Class-3S = value-is-better
Class-4S = range-is-better

Hard:

Class-1H = must be smaller, i.e., $g_i \leq g_{i_{\max}}$
Class-2H = must be larger, i.e., $g_i \geq g_{i_{\min}}$
Class-3H = must be equal, i.e., $g_i = g_{i_{\text{val}}}$
Class-4H = must be in range, i.e., $g_{i_{\min}} \leq g_i \leq g_{i_{\max}}$

Within conventional design optimization approaches, the metric for which class 1S or 2S applies would generally become part of

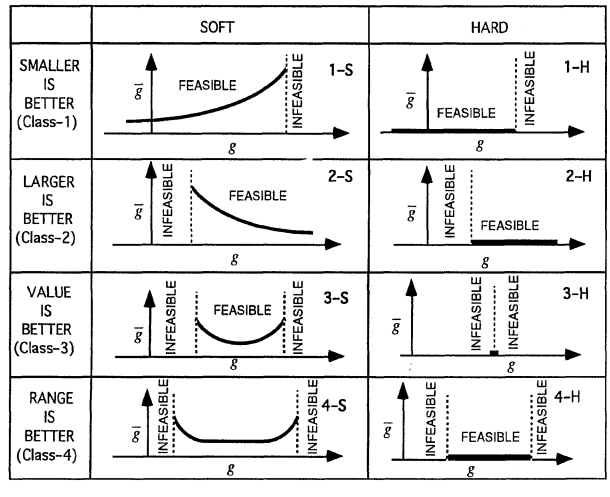


Fig. 1 Classification of design metrics.

the aggregate objective function, with a multiplicative weight, and all the hard classes would become constraints. Cases of class 3S and 4S would be significantly more difficult to treat. One approach would be to use a positive or negative weight, depending on the current value of the corresponding metric during optimization. A large amount of trial and error would be involved in choosing the right weights. Physical programming removes this trial and error entirely by using the class functions.

The class functions shown in Fig. 1 provide the means for a designer to express a range of preferences for a given design metric. As shown in Fig. 1, the soft class functions provide information that is deliberately imprecise. By design, the utopian value of the class functions is zero. Next, we explain how quantitative specifications are associated with each design metric. (Note that class 4S actually represents an extension of the work in Ref. 1.)

4. Physical Programming Lexicon

Physical programming allows the designer to express preferences with regard to each metric with more specificity than by simply saying minimize, maximize, greater than, less than, or equal to. Physical programming explicitly recognizes the limitations of such a problem formulation framework and addresses these by employing a flexible lexicon. This lexicon comprises terms that characterize the degree of desirability of 6 ranges for each generic design metric for classes 1-S and 2-S, 10 ranges for classes 3-S, and 11 ranges for class 4-S. To illustrate, consider the case of class 1-S (Fig. 2). The ranges are defined as follows, in order of decreasing preference.

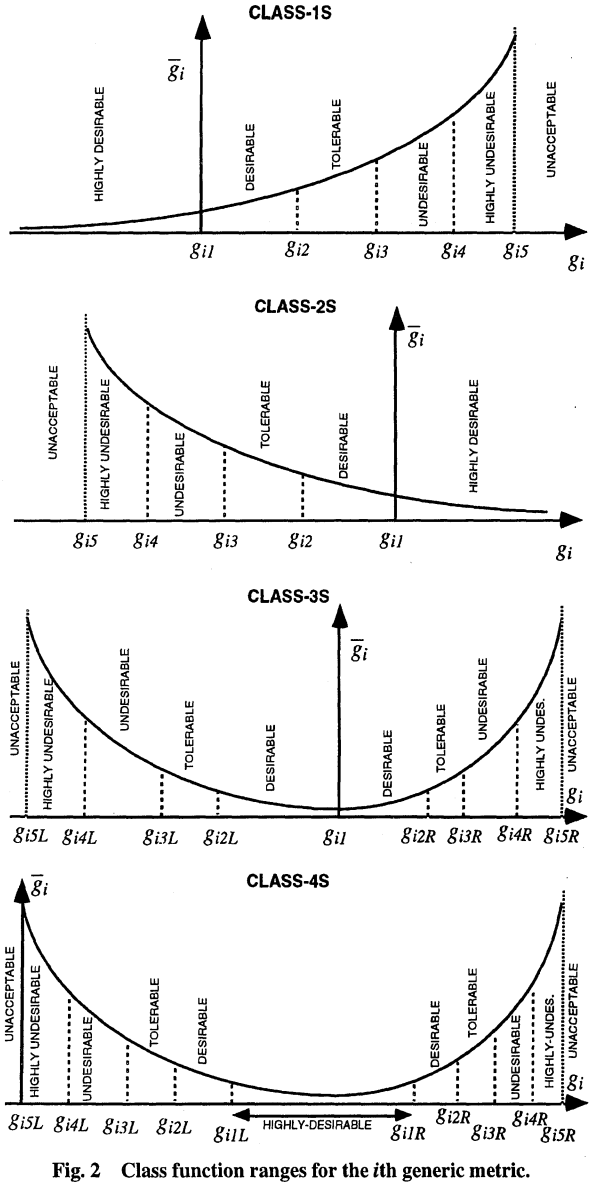


Fig. 2 Class function ranges for the *i*th generic metric.

1) Highly desirable range ($g_i \leq g_{i1}$): an acceptable range over which the improvement that results from reduction of the design metric is desired but is of minimal additional value.

2) Desirable range ($g_{i1} \leq g_i \leq g_{i2}$): an acceptable range that is desirable. A smaller value of the design metric, over this range, is preferred over a larger value thereof.

3) Tolerable range ($g_{i2} \leq g_i \leq g_{i3}$): an acceptable, tolerable range. A smaller value of the design metric, over this range, is preferred over a larger value thereof.

4) Undesirable range ($g_{i3} \leq g_i \leq g_{i4}$): a range that, although acceptable, is undesirable. A smaller value of the design metric, over this range, is preferred over a larger value thereof.

5) Highly undesirable range ($g_{i4} \leq g_i \leq g_{i5}$): a range that, although still acceptable, is highly undesirable. A smaller value of the design metric, over this range, is preferred over a larger value thereof.

6) Unacceptable range ($g_i \geq g_{i5}$): The range of values that the generic metric must not take. All design metric values in this range lead to unacceptable designs.

We note that g_{i1}–g_{i5} are (physically meaningful) values of the design metric g_i that are prescribed by the designer to quantify the preferences associated with the ith metric. These prescribed values delineate the desirability ranges within each metric. Reference 1 discusses the quantitative implications of the preceding definitions and provides the mathematical procedure for the development of the class function for each generic metric.

The class functions map design metrics, such as t_s, into nondimensional, strictly positive real numbers. This mapping, in effect, transforms metrics with disparate units and physical meaning onto a dimensionless scale through a unimodal function. Figure 2 illustrates the mathematical nature of the class functions and shows how they allow the designer to express the range of goodness, or preferences, of a given metric. Consider the first curve of Fig. 2, the class function for class 1S design metrics. Six ranges are defined. The parameters g_{i1}–g_{i5} are specified by the designer. When the value of the metric g_i is less than g_{i1} (highly desirable range), the value of the class function is small, which requires little further minimization of the class function. When, on the other hand, the value of the metric g_i is between g_{i4} and g_{i5} (highly undesirable range), the value of the class function is large, which requires significant minimization of the class function. The behavior of the other class functions is indicated in the figure. Preferences regarding each metric are treated independently, allowing the inherent multiobjective nature of the problem to be preserved. This describes the basic process of physical programming: the value of the class function for each metric governs the optimization path in objective space (metric space).

B. Physical Programming Mappings

We now briefly discuss the various mappings that take place in the implementation of physical programming. These mappings define the path from design variables to the aggregate objective function, which is the actual function that the nonlinear programming code minimizes. Figure 3 shows these various mappings. As illustrated in Fig. 3, we begin with the design variables $\boldsymbol{x}$ over which the designer has control, e.g., location of poles and zeros in a controller transfer function. The design variables are mapped into the objective function space $\boldsymbol{g}$ using the metrics evaluation module (MEM) (see Sec. II.C). The numerical contribution of a given metric g_i to the aggregate objective function depends 1) on the value of the metric, e.g., a particular t_s value that results from a particular set of design parameter values; 2) on the class type assigned to the given design metric, e.g., smaller-is-better; and 3) on the assigned range values associated with the metric (g_{i1}–g_{i5}). Loosely speaking, the sum of all the class functions that represent mappings of the design metric equals the aggregate objective function (see Ref. 1 for details). We note, however, that the class functions are not formed independently.

C. Physical Programming Application Steps

With this background in physical programming terminology and mapping, we are prepared to explain the procedure for solving a problem with physical programming.

Step 1. Create a software MEM that uses the current numerical values of the design parameters $\boldsymbol{x}$ as input and provides the

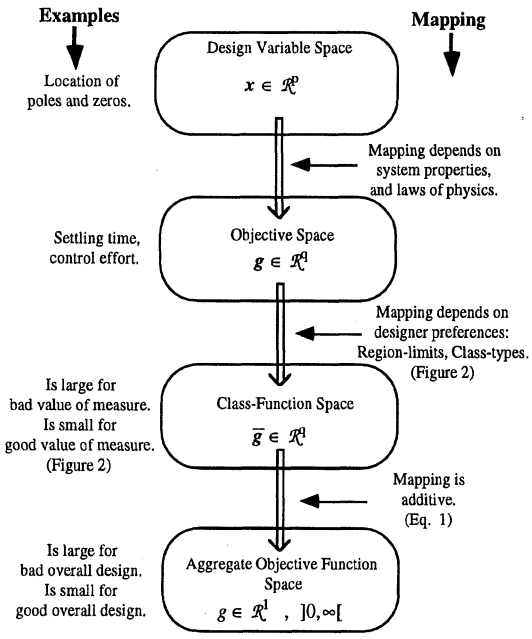


Fig. 3 Physical programming mappings.

corresponding numerical values for the vector of design metrics $\boldsymbol{g}$ as output. This module models the behavior of the physical system. In the present case, because PhysPro is implemented in the Matlab environment, the MEM development can exploit the full power and versatility of Matlab, or can also be a C or Fortran module that interfaces with Matlab.

Step 2. Specify the class type for each metric (1S–4H); see Fig. 1.

Step 3. Define the range limits for each metric; see Fig. 2. The designer specifies 5 limits for classes 1S or 2S, 9 for 3S, and 10 for 4S. For the hard classes, the designer specifies one limit for classes 1H, 2H, or 3H and two limits for 4H (see Fig. 1). The set of these range limits recorded in tabular form constitutes the preference table.

Step 4. Solve the constrained minimization that is defined by Eq. (1). Note that, if a physical programming software implementation is available to the designer, steps 1–3 constitute the extent of the designer's involvement with nonlinear programming. Rather than engaging in the obscure and dubious path of tweaking weights, the designer's time can be constructively spent exploring the implications of the various physically meaningful preference choices.

D. Physical Programming Problem Model

The physical programming problem model takes the form

$$\min_x g(x) = \min_x \log_{10}\left\{\frac{1}{n_{sc}}\sum_{i=1}^{n_{sc}} \bar{g}_i[g_i(x)]\right\} \qquad \text{(for soft classes)} \tag{1}$$

subject to

$$g_i(x) \le g_{i5} \qquad \text{(for class 1S metrics)}$$

$$g_i(x) \ge g_{i5} \qquad \text{(for class 2S metrics)}$$

$$g_{i5L} \le g_i(x) \le g_{i5R} \qquad \text{(for class 3S metrics)}$$

$$g_{i5L} \le g_i(x) \le g_{i5R} \qquad \text{(for class 4S metrics)}$$

$$g_i(x) \le g_{iM} \qquad \text{(for class 1H metrics)}$$

$$g_i(x) \ge g_{im} \qquad \text{(for class 2H metrics)}$$

$$g_i(x) = g_{iv} \qquad \text{(for class 3H metrics)}$$

$$g_{im} \le g_i(x) \le g_{iM} \qquad \text{(for class 4H metrics)}$$

$$x_{jm} \le x_j \le x_{jM} \qquad \text{(for des. var. constraints)}$$

where g_{im}, g_{iM}, x_{jm}, and x_{jM} represent minimum and maximum values, and the various g_{iv} define the equality constraints. The range limits are provided by the designer (see Tables 1–3 and Fig. 2), and n_{sc} is the number of soft metrics that the problem comprises. Note that the aggregate objective function only comprises class functions associated with soft metrics. The hard metrics are treated as constraints.

Table 1 Preference table for (2, 3) controller

Design metric:	Highly desirable	Desirable	Tolerable	Undesirable	Highly undesirable
	g_{i1}	g_{i2}	g_{i3}	g_{i4}	g_{i5}
$Re(\lambda_i)^{\text{cl-loop}}_{\text{max-}i,k}$	−0.001	−0.0005	−0.0001	−0.00005	−0.00001
$u^{\text{max-}t}_{\text{max-}k}$	7	8	8.5	9	2
$t_{s\,\text{max-}k}$	1000	2000	3000	4000	5000
$\text{Noise}_{\text{max-}k}$	5	10	30	40	50
$u^{\text{max-}t}_{\text{nom}}$	1	2	3	4	5
$t_{s\,\text{nom}}$	13.5	14	14.5	17	21

Table 2 Preference table for (3, 4) controller

Design metric:	Highly desirable	Desirable	Tolerable	Undesirable	Highly undesirable
	g_{i1}	g_{i2}	g_{i3}	g_{i4}	g_{i5}
$Re(\lambda_i)^{\text{cl-loop}}_{\text{max-}i,k}$	−0.01	−0.005	−0.001	−0.0005	−0.00001
$u^{\text{max-}t}_{\text{max-}k}$	0.80	0.85	0.95	1.0	2
$t_{s\,\text{max-}k}$	15	40	80	90	100
$\text{Noise}_{\text{max-}k}$	1.8	2.0	2.2	2.5	3.0
$u^{\text{max-}t}_{\text{nom}}$	0.9	1.2	2.0	2.5	3.0
$t_{s\,\text{nom}}$	14	14.2	14.4	14.6	15

Table 3 Preference table for (4, 5) controller

Design metric:	Highly desirable	Desirable	Tolerable	Undesirable	Highly undesirable
	g_{i1}	g_{i2}	g_{i3}	g_{i4}	g_{i5}
$Re(\lambda_i)^{\text{cl-loop}}_{\text{max-}i,k}$	−0.01	−0.005	−0.001	−0.0005	−0.0001
$u^{\text{max-}t}_{\text{max-}k}$	0.85	0.90	1.0	1.5	2.0
$t_{s\,\text{max-}k}$	14	16	18	21	25
$\text{Noise}_{\text{max-}k}$	0.5	0.9	1.2	1.4	1.5
$u^{\text{max-}t}_{\text{nom}}$	0.5	0.7	1.0	1.5	2.0
$t_{s\,\text{nom}}$	10	11	12	14	25

The preceding problem model conforms to the framework of most nonlinear programming codes, with possible minor rearrangements. However, note that the form of the aggregate objective function departs markedly from conventional design optimization methods. We now apply the preceding development to the two-mass benchmark problem.

III. Benchmark Problem and Specification

The benchmark problem consists of the mass–spring–mass system in Fig. 4, which represents a generic model of an uncertain dynamical system with one rigid-body mode and one vibrational mode.[11] The nominal values of the first mass m_1, the second mass m_2, and the linear stiffness k are unity. A control force u acts on body 1 while the position of body 2 is measured, resulting in a noncollocated control problem. Body 1 and body 2 are assumed to experience the respective disturbances w_1 and w_2. The variables x_1 and x_2 represent the respective displacements of body 1 and body 2; y is the measurement of x_2; v is the sensor noise; and $z(= x_2)$ is the controlled output quantity.

The system is represented in state-space form as

$$\begin{Bmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\\ \dot{x}_4\end{Bmatrix} = \begin{bmatrix}0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\\ -k/m_1 & k/m_1 & 0 & 0\\ k/m_2 & -k/m_2 & 0 & 0\end{bmatrix}\begin{Bmatrix}x_1\\ x_2\\ x_3\\ x_4\end{Bmatrix} + \begin{bmatrix}0\\ 0\\ 1/m_1\\ 0\end{bmatrix}(u + w_1) + \begin{bmatrix}0\\ 0\\ 0\\ 1/m_2\end{bmatrix} w_2 \tag{2}$$

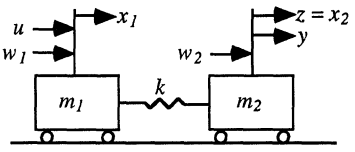


Fig. 4 Coupled mass benchmark problem.

$$y = x_2 + v \tag{3}$$

$$z = x_2 \tag{4}$$

This paper addresses the benchmark problem 1 (Ref. 11). The objective is to design a controller whose closed-loop performance meets six requirements, which we repeat here with clarification for our interpretation.

1) For a unit impulse disturbance exerted on body 1 and/or body 2, the controlled output ($z = x_2$) has a settling time of about 15 s for the nominal system ($m_1 = m_2 = k = 1$). We applied a disturbance only on the second mass, as this was the condition used by Jayasuriya et al.[14] and Wie et al.,[15] with which we compare our results.

2) The closed-loop system is stable for $0.5 \leq k \leq 2.0$ and $m_1 = m_2 = 1$.

3) The closed-loop system is insensitive to high-frequency sensor noise. We designed only strictly proper controllers and evaluated the H_∞ norm of the noise response (see Sec. IV for details).

4) Reasonable performance and stability robustness and reasonable gain and phase margins are achieved with reasonable bandwidth. We did not evaluate worst-case gain and phase margins. Stengel and Marrison[10] have shown that these can be poor indicators of robust performance. We instead determined the worst-case stability margin, as measured by $\max\{Re(\lambda[A])\}$, where A is the closed-loop system matrix.

5) Reasonable control effort, e.g., peak control input $u_{\max}$, is used. We determined $u_{\max}$ for both nominal and worst-case values of the stiffness k.

6) Reasonable controller complexity is sought. (We used controller order as an indicator of controller complexity.)

We chose this interpretation because it allows for comparison with Wie et al.[15] and Jayasuriya et al.,[14] which we do in the sequel.

IV. Method of Investigation

We determined the suitability of a given controller by its ability to meet the six objectives of the two-mass benchmark problem. In accordance with these objectives and our interpretation, we created several algorithms that provide design metric values corresponding to a given controller and uncertain plant. A description of these metrics and algorithms follows.

Settling time: Identify the maximum time t_s when $|x_2(t_s)| \geq 0.1$ in response to a unit impulse disturbance. This time is denoted $t_{s\,\text{max-}k}$. We also consider the nominal settling time $t_{s\,\text{nom}}$, defined for stiffness $k = 1.0$.

Stability: Ensure that

$$Re(\lambda_i)^{\text{cl-loop}}_{\text{max-}i,k} = \max_{k \in [0.5,\, 2.0]} Re\{\lambda[A(k)]\} < 0 \tag{5}$$

where $A(k)$ is the system matrix and k the stiffness constant. Ideally, this metric is not needed. If $u^{\text{max-}t}_{\text{max-}k}$ is finite, the preceding condition is automatically satisfied. In practice, however, the computation of $u^{\text{max-}t}_{\text{max-}k}$ could result in a finite number even in the presence of an unstable mode if that mode's eigenvalue has a very small positive real part. The inclusion of $Re(\lambda_i)^{\text{cl-loop}}_{\text{max-}i,k}$ is therefore warranted.

Sensitivity: Use only a strictly proper controller and determine the ratio of

$$\text{Noise}_{\text{max-}k} = \max_{\substack{k \in [0.5,\, 2.0] \\ \omega \in [100,\, 10{,}000]}} \left| \frac{u(j\omega)}{v(j\omega)} \right| \Big/ \left| \frac{1}{j\omega} \right| \tag{6}$$

where u is the output of the controller and v is sensor noise input. The sensitivity gives an indication of the sensor noise amplification vs a first-order rolloff over a specific frequency range.

Control effort: Determine the maximum control effort (output of controller) in response to an impulse disturbance. It is denoted $u^{\text{max-}t}_{\text{max-}k}$. We also consider the nominal control effort $u^{\text{max-}t}_{\text{nom}}$.

Controller complexity: The controller denominator order specifies controller complexity.

Next we created the MEM using Matlab. The MEM takes the controller numerator and denominator coefficients as input and uses the preceding algorithms to return the six design metrics, as described earlier. We used the MEM to evaluate the performance of designs found in the literature and to obtain our own physical-programming-based designs. Our investigation did not involve a comprehensive attempt to improve upon every solution in the literature, nor did it seek to identify specific strengths or deficiencies of particular controllers. Rather, we used representative published solutions as a reference point to facilitate discussions regarding the effectiveness of physical programming as a control design tool.

We designed three physical-programming-based controllers of the form

$$G^{mn}_{pp}(s) = \frac{\left(\sum_{i=0}^{m} b_i s^i\right)}{\left(\sum_{j=0}^{n} a_j s^j\right)} \tag{7}$$

for the cases $(m, n) = (2, 3), (3, 4)$, and $(4, 5)$, respectively referred to as PP23, PP34, and PP45. We note that all controllers are configured with positive feedback. The coefficients a_i and b_i are the design parameters. We then compared the performance obtained with these controllers against published designs for each controller-order case. The designs with which we compared our results are Byrns et al.[16] for the (2, 3) case, Wie et al.[15] for the (3, 4) case, and Jayasuriya et al.[14] for the (4, 5) case. The transfer functions of these controllers are as follows.

Byrns (2, 3) (B23):

$$\text{B23} = \frac{-40.42(s + 2.388)(s + 0.350)}{(s + 163.77)(s^2 + 2 \times 0.501 \times 0.924s + 0.924^2)} \tag{8}$$

Wie (3, 4) (W34):

$$\text{W34} = \frac{2.1303(s + 3.434)(s - 0.984)(s + 0.145)}{(s^2 + 2 \times 0.459 \times 2.24s + 2.24^2)(s^2 + 2 \times 0.825 \times 1.586s + 1.586^2)} \tag{9}$$

Jayasuriya (4, 5) (J45):

$$\text{J45} = \frac{-3.047 \times 10^5 (s + 0.66)(s + 2.24)(s^2 + 2 \times 0.355 \times 0.813s + 0.813^2)}{(s + 6.66)(s^2 + 2 \times 0.925 \times 5.877s + 5.877)(s^2 + 2 \times 0.375 \times 40s + 40^2)} \tag{10}$$

We did not design controllers of order lower than the (2, 3) case because this controller was the lowest order that even approximated solving the two-mass benchmark problem. The other two controllers produced satisfactory designs, and because reasonable controller order is one of the requirements of the benchmark problem, we chose not to investigate higher-order designs.

V. Results

As discussed in Sec. II, physical-programming-based design involves the creation of a preference table that defines a designer's preference with respect to each design metric, e.g., settling time. Although the requirements are the same regardless of the controller

Table 4 Comparing results from physical programming and other controllers

Controller	$Re(\lambda_i)^{\text{cl-loop}}_{\text{max-}i,k}$	$u^{\text{max-}t}_{\text{max-}k}$	$t_{s\ \text{max-}k}$	$\text{Noise}_{\text{max-}k}$	$u^{\text{max-}t}_{\text{nom}}$	$t_{s\,\text{nom}}$**
PP23	**−0.0025**	**0.564**	**21.5**	**12.5**	**0.455**	**20.7**
B23	0.14*	∞	∞	∞	0.513	20.9
PP34	**−0.054**	**0.668**	**18.4**	**0.662**	**0.522**	**13.5**
W34	−0.035	0.706	30.9	2.13	0.573	14.8
PP45	−0.0168	**0.589**	18.4	**1.14**	**0.462**	12.1
J45	**−0.227**	145	**6.61**	340,000	121	**6.41**

*Robustly unstable controller.
**Required maximum settling time is 15 s.

order, we created different preference tables for each controller (see Tables 1–3). We did this because, as expected, we can usually be more demanding of a high-order controller than we can be of a low-order controller. In fact, after exploring the design space, we could not satisfy the 15-s settling time requirement after several attempts for the (2, 3) controller. Thus, to obtain a feasible solution, we intentionally chose to request a higher 21-s settling time requirement for this controller, which otherwise performed well. Conversely, a different situation prevails for PP34 and PP45 controller preferences. Observe, for example, that in the PP34 case (see Tables 1–3) a nominal settling time of less than 14 s is considered highly desirable, whereas in the PP45 case a nominal settling time that is highly desirable could be set at less than 10 s; i.e., the higher-order controller enables us to specify a more demanding set of preferences. Note that, because we intend to minimize all of the designs metrics described earlier, class 1S applies for all the metrics; thus the designer must specify five constants for each design metric (Tables 1–3).

Using physical programming, we were able to solve the two-mass benchmark problem with both the (3, 4) and (4, 5) controllers; however, the low order of the (2, 3) controller did not provide sufficient loop-shaping flexibility to achieve the 15-s settling time. For the (2, 3) case, the PP23 controller robustly stabilized the mass–spring–mass system and achieved a worst-case t_s of 21.5 s, approximately 43% higher than the nominal design objective. The B23 controller did not robustly stabilize the system; hence, its worst-case t_s is infinite.

The PP23 controller transfer function has the form

$$G^{23}_{pp}(s) \equiv \text{PP23} = \frac{12.5(s-1.23)(s+0.203)}{(s+20.6)(s^2+2\times 0.501\times 1.21s+1.21^2)} \tag{11}$$

Both the PP34 and the PP45 controllers resulted in closed-loop response that met all of the objectives of the two-mass benchmark problem. Furthermore, the PP45 controller significantly improved on results from the literature (J45). The PP34 transfer function has the form

$$G^{34}_{pp}(s) \equiv \text{PP34} = \frac{0.66(s-1.07)(s+0.124)(s+7.16)}{(s^2+2\times 0.304\times 2.38s+2.38^2)(s^2+2\times 0.769\times 1.28s+1.28^2)} \tag{12}$$

and the PP45 transfer function has the form

$$G^{45}_{pp}(s) \equiv \text{PP45} = \frac{1.14(s-1.11)(s+0.0169)(s+0.112)(s+6.43)}{(s+0.0016)(s+1.28)(s+2.11)(s^2+2\times 0.456\times 2.28s+2.28^2)} \tag{13}$$

Although both the PP34 and the PP45 controllers resulted in improved performance, as compared with the cited results, our main purpose here is to demonstrate physical programming's ability to solve the two-mass benchmark problem. The superiority of the physical programming controllers over other baseline controllers supports this claim and demonstrates physical programming's ability to provide significant aid in multiobjective computational controller design. The performance of all of the controllers is presented in Table 4.

VI. Discussion

Physical programming's ability to balance conflicting objectives leads to controllers that perform well with respect to all aspects of the benchmark problem. To see this, we analyze the system response obtained from the PP23, PP34, and PP45 controllers and from the B23, W34, and J45 controllers. We also evaluate all of the controllers in terms of their respective design metrics, shown in Table 4, where the numbers in bold are lower and indicate better performance for a given controller order.

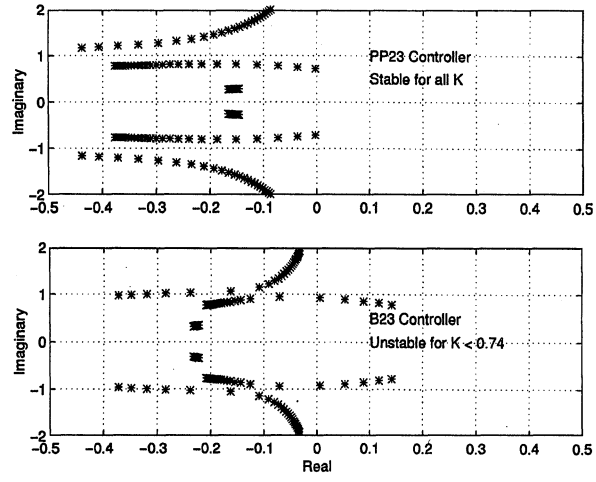


Fig. 5 Root loci for physical programming and B23 controllers.

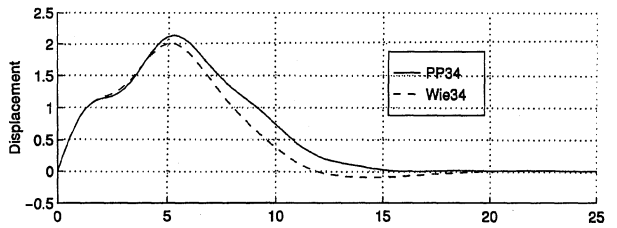
A. (2, 3) Controllers

Neither the physical programming controller PP23 nor the B23 controller met the 15-s settling time specification with the nominal model. Attempts to meet this requirement by adjusting the preference table did not lead to a successful physical programming design. The settling time specification was not met because the lower order of this controller provided significantly less capability in terms of loop shaping. As noted in the result section, the PP23 controller is robustly stable, whereas the B23 controller is unstable at low values of the spring stiffness (see Table 4). Figure 5 shows the root locus of each (2, 3) controller; we see that the B23 controller has two branches that cross the imaginary axis, whereas the PP23 controller shows robustly stable behavior. Table 4 shows that these two controllers otherwise display similar performance. The preference table for the PP23 controller is shown in Table 1. Note that, in an effort to try to satisfy the settling time requirement, the worst-case settling time is set to arbitrarily large values to prevent that metric from impacting the solution. Thus, this metric's preference is automatically satisfied.

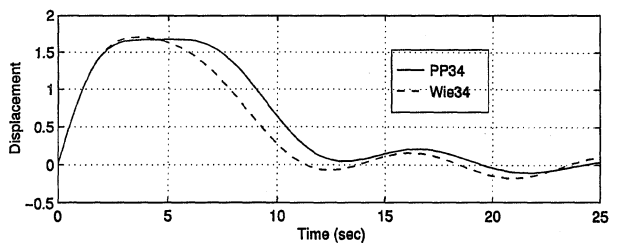
B. (3, 4) Controllers

The controllers W34 and PP34 each met the settling time specification and performed generally well. In Fig. 6, we plot nominal and worst-case (in terms of k) displacement of the second mass, in response to an impulse disturbance at the same mass. For the nominal case, the physical programming closed-loop response displays negligible undershoot following the peak displacement of the mass. The W34 controller has a slight undershoot, which leads to a minutely longer settling time. Worst-case settling time resulted when $k = 0.5$ for both the physical programming controller and the Wie controller. Figure 6 shows that they both display similar dominant behavior, to which lightly damped second-order responses are superimposed. The amplitude of that second-order response is lower in the case of physical programming. The worst-case t_s from the W34 controller is approximately 68% greater than in the case of physical programming controller (Table 4).

In Fig. 7, we plot nominal and worst-case actuator efforts, as before, in response to an impulse disturbance on the second mass. The actuator efforts (the output of the controller) for the PP34 and

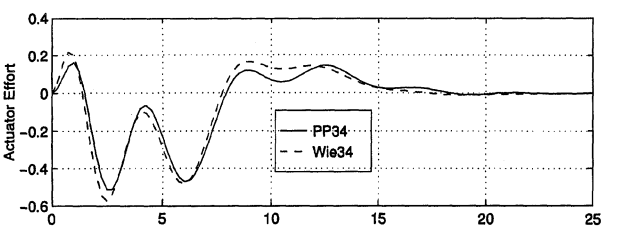


Nominal stiffness (k = 1.0)

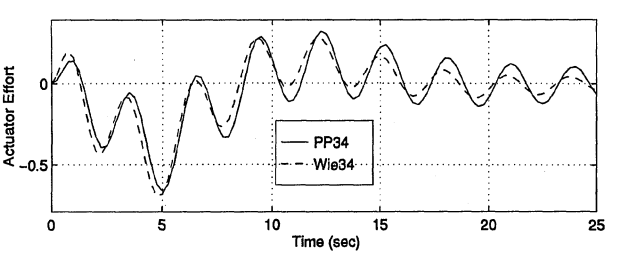


Worst-case stiffness (k = 0.5)

Fig. 6 Impulse responses for physical programming and W34 controllers.



Nominal stiffness (k = 1.0)



Worst-case stiffness (k = 2.0)

Fig. 7 Actuator effort for physical programming and W34 controllers.

W34 controllers are similar in both the nominal and worst cases. However, in each case, the PP34 controller requires less effort than the W34 controller.

C. (4, 5) Controllers

As in the case of the (3, 4) controllers, both the physical programming (PP45) and the Jayasuriya et al. (J45) controllers met the settling time specification. However, the controllers differ markedly in several other important respects. The nominal and maximum control efforts of the physical programming controller are two orders of magnitude smaller than those demanded by the J45 controller. Furthermore, peak noise amplification of the physical programming controller is five orders of magnitude less than the other controller. The nominal settling time of the J45 controller is 6.41 s, approximately one-half of the physical programming controller's settling time. The worst-case settling-time, 6.61 s, is about one-third the physical programming controller's. The shorter settling time of the J45 controller results from a higher closed-loop bandwidth. However, this bandwidth is achieved at the expense of a very large controller effort and a very large noise amplification.

VII. Summary

Physical programming's ability to incorporate and (when possible) satisfy multiple conflicting objectives leads to a controller that meets all of the aspects of the benchmark problem: nominal and worst-case settling time, nominal and worst-case controller effort, and noise amplification. Furthermore, physical programming meets these objectives without excessively sacrificing one objective in favor of another. The preceding comparisons demonstrate physical programming's ability to weigh tradeoffs (settling time vs control effort vs noise amplification) and to accommodate these while designing controllers that lead to improved overall performance. Furthermore, these improvements are obtained by directly adjusting preferences expressed in terms of the physically meaningful design metrics. Using other techniques, improved performance is obtained indirectly, e.g., weights in a matrix or frequency-dependent terms are adjusted by trial and error.

Finally, we provide a final comment regarding the preference table. A prospective user of physical programming might wonder how tedious it is to set the preference table. Indeed, setting the preference table often does involve an iterative process. However, this iterative process of prescribing preference is distinct from the trial and error nature of setting weights. The former takes place in a physically meaningful design environment, whereas the latter provides little guidance to designers as they navigate through design space. When we are not satisfied with the outcome resulting from a highly desirable settling time set to 14 s, we may reset this preference value to 11 or 12 s. On the other hand, when we use weight methods, if a weight of 100 does not lead to a good design, we have no idea whether to increase the weight to 150, 500, or 1000. Furthermore, if we wish to alter the behavior of several design metrics of the system, we need to change several weights simultaneously, thereby further exacerbating the difficulties. Reference 1 addresses this issue.

The versatility of physical programming can also be seen in Ref. 17, where the construction of the generic aggregate objective function is discussed, and in Ref. 18, where the control structure integrated design problem is addressed. In the latter, more computationally practical methods for design metric evaluation are presented.

VIII. Concluding Remarks

In this paper, we have provided a demonstration of physical programming's ability to synthesize a controller that solves the two-mass benchmark problem. While coping with multiple conflicting objectives, physical programming solved the benchmark problem directly and explicitly. Physical programming effectively brings the benefits of nonlinear programming within the reach of the control designers who are more adept at designing controllers than at coping with the intricacies of numerical optimization. The results in this paper strongly suggest that more work in physical programming control synthesis should be undertaken to investigate its effectiveness as a general approach to control synthesis.

Acknowledgments

Support by the National Science Foundation, Grants CMS-9622652 and CMS-9410144, is gratefully acknowledged.

References

[1]Messac, A., "Physical Programming: Effective Optimization for Computational Design," *AIAA Journal*, Vol. 34, No. 1, 1996, pp. 149–158.

[2]Messac, A., and Hatttis, P., "Physical Programming Application to the High Speed Civil Transport Plane Design," *Journal of Aircraft*, Vol. 33, No. 2, 1996, pp. 446–449.

[3]Messac, A., "Optimal Simultaneous Structural and Control Optimization of Large Space Structures," Ph.D. Dissertation, Dept. of Aeronautics and Astronautics, Massachusetts Inst. of Technology, Cambridge, MA, Nov. 1985.

[4]Onoda, J., and Haftka, R. T., "An Approach to Structure/Control Simultaneous Optimization for Large Flexible Spacecraft," *AIAA Journal*, Vol. 25, No. 8, 1987, pp. 1133–1138.

[5]Rao, S. S., Venkaya, V. B., and Khot, N. S., "Game Theory Approach for the Integrated Design of Structures and Controls," *AIAA Journal*, Vol. 26, No. 4, 1988, pp. 463–469.

[6]Messac, A., and Malek, K., "Control Structure Integrated Design," *AIAA Journal*, Vol. 30, No. 8, 1992, pp. 2124–2131.

[7]Tabak, D., and Kuo, B., C., *Optimal Control by Mathematical Programming*, Prentice–Hall, Englewood Cliffs, NJ, 1971.

[8]Bulirsch, R., and Kraft, D. (eds.), *Computational Optimal Control*, Vol. 155, ISNM International Series of Numerical Mathematics, Birhäuser Verlag, Boston, MA, 1994.

[9]Wie, B., and Bernstein, D., "A Benchmark Problem for Robust Control Design," *Proceedings of the American Control Conference*, Vol. 3, Chicago, IL, 1992, pp. 2047, 2048.

[10]Stengel, F. R., and Marrison, C., "Robustness of Solutions to a Benchmark Control Problem," *Journal of Guidance, Control, and Dynamics*, Vol. 15, No. 5, 1992, pp. 1060–1067.
[11]Alfriend, K. T., "Robust Control Design for a Benchmark Problem," *Journal of Guidance, Control, and Dynamics*, Vol. 15, No. 5, 1992, p. 1057.
[12]"PhysPro: A Physical Programming Software Package for Optimal Design," Optimal Systems, Lexington, MA, 1997.
[13]"Matlab," The MathWorks, Inc., Natick, MA, 1997.
[14]Jayasuriya, S., Yaniv, O., Nwokah, O. D. I., and Chait, Y., "Benchmark Problem Solution by Quantitative Feedback Theory," *Journal of Guidance, Control, and Dynamics*, Vol. 15, No. 5, 1992, pp. 1087–1093.
[15]Wie, B., Liu, Q., and Byun, K.-W., "Robust H_∞ Control Synthesis Method and Its Application to Benchmark Problem," *Journal of Guidance, Control, and Dynamics*, Vol. 15, No. 5, 1992, pp. 1140–1148.
[16]Byrns, E. V., and Calise, A. J., "Fixed-Order Dynamic Comensators for the H_2/H_∞ Benchmark Problem," *Proceedings of the 1990 American Control Conference*, Vol. 3, San Diego, CA, 1990, pp. 963–965.
[17]Messac, A., "From the Dubious Art of Constructing Objective Functions to the Application of Physical Programming," AIAA Paper 96-4123, April 1996.
[18]Messac, A., and Akbulut, B., "Control-Structure Integrated Design with Closed-Form Design Metrics Using Physical Programming," AIAA Paper 97-1287, April 1997.

A. D. Belegundu
Associate Editor