Applying a generic constraint solving technique to engineering design

Hiroyuki Sawada [1] and **Xiu-Tian Yan** [2]

Abstract. As a design solution becomes more concrete and detailed during engineering design, more design parameters are introduced to specify the solution quantitatively. This increased number of design parameters causes challenges and difficulties to a designer in terms of gaining an insight into the significance of these design parameters and their influence on the overall performance of a product. This in turn can easily lead a designer to generating a less optimal final design solution. In order to overcome such difficulties, a design support system based on a generic constraint solving technique is derived to support engineering design. The system is concerned with three types of constraints, namely kinematic, energetic and spatial constraints of a product. Standard components can be selected from the system component library and a product can be subsequently configured using these components. The system then can automatically generate all the related constraints and a design solution can be selected from solutions obtained from constraint problem solving. The system can also deal with an incompletely defined design solution.

1 Introduction — Preliminary Design Process —

In engineering design, particularly in mechatronic system design, when a new artifact is designed based on customer's design specification, a designer typically goes through a number of design process stages: namely problem analysis, conceptual design, embodiment design and detail design [3, 6]. Design usually commences with a need. The purpose of the problem analysis stage is to translate this need in customer's description into a Product Design Specification [7]. This can then be used as a target document for both generating a solution and measuring it. At conceptual design stage, a number of alternative solution concepts [6] or schemes [3] are normally generated for further exploration. Embodiment design stage is concerned with refining relatively abstract description of concepts into more concrete and definitive solution description. More quantitative design parameters are introduced at this stage to make a solution more concrete. Constraints to be satisfied by the new product design parameters can be described in the form of equation and inequality. Those design parameters associated with their constraints may be of kinematic, energetic and spatial nature of the product. The key activity here is the description of constraints. At detail design stage, the scheme selected at the embodiment design stage is worked out in greater details, resulting in documentation traditionally in the form of assembly drawings, detail drawings and parts lists. During this stage, refining and optimizing the definitive solution description are the key tasks and these can be done by computing parameter values satisfying all constraints. This stage is hence essentially a stage of computation of parameter values.

Of interest to this paper are last two stages of a design process, description of constraints and computation of design parameter values. There exist significant problems during these two activities.

(a) Problems about description of constraints
In innovative design, constraints are often described based on a freehand sketch drawn by a designer. For example, when a new robot arm is designed, constraints are defined by referring to a sketch in which an arm is drawn as a segment and a force vector is expressed as an arrow. In such a method, designer's work load involving robot arm's kinematic, energetic and spatial constraints is heavy and it is difficult to avoid designer's mistakes caused by misunderstanding of the sketch and the complexity of the problem.

(b) Problems about computation of parameter values
Usually, parameter values are computed by executing computer programs made by a designer based on a set of constraints. These values are determined using different analysis programs from different viewpoints. A different analysis of a design solution requires a different computer program. For example, kinematic analysis uses a different program from system function analysis. Furthermore, though it is necessary to assign concrete values to all the input parameters in a conventional computer program, several parameters are left unknown at the early design stage. Using a conventional method, it is difficult to handle a formula that contains unknown input parameters.

Furthermore, when an alternative design solution to the same design requirements is considered, the above process has to be repeated. For example, since both kinematic and force balance relationships about two-joint and three-joint arm robots are completely different, constraint sets and computer programs are also different. Since the computation is complex and the workload is very intense, a designer often accepts the first design solution without exploring alternative design solutions and finding optimal one.

This research aims to overcome these difficulties described above by applying a generic constraint solving technique, and to develop a design support computer system to reduce designer's work load. Furthermore, this system enables a designer to explore alternative design solutions to complex mechatronic design problems.

The structure of this paper is as follows. In Section 2, approaches of this research are described. Section 3 explains the architecture of the design support computer system. In Section 4, a design problem of a two-fingered-two-joint robot is shown as an example.

[1] Mechanical Engineering Laboratory, Namiki 1–2, Tsukuba, Ibaraki 305–8564, Japan, email: sawada@mel.go.jp
[2] University of Strathclyde, James Weir Building, 75 Montrose Street, Glasgow G1 1XJ, UK, email: x.t.yan@dmem.strath.ac.uk

2 Approaches of this research

This section describes a complementary set of approaches to two problems described in the previous section.

2.1 Component library — Approach to description of constraints —

In engineering design, a designer seldom generates a design solution from scratch. A new design solution of conceptual design is typically presented as a new structure constructed by combining available basic components such as links, gears and motors. From the viewpoint of defining constraints, this is regarded as a process in which a global constraint set about the product is composed by combining local constraint sets about basic components and adding interface conditions between those components.

The authors propose to provide a designer with a database that has constraint sets about basic components commonly used in engineering design. The database is called the *component library*. A designer selects necessary components from it and obtains a global constraint set about the product by adding interface conditions between these components. Figure 1 shows a link as an example of components in

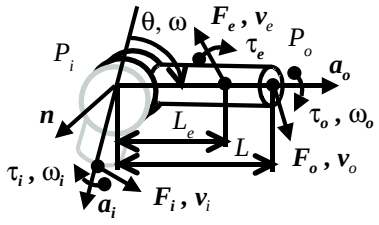


Figure 1. A component link representation

the component library. The constraints are as below.

Constraints about the link

Kinematic constraints

1. The direction vectors and the rotational axis vector have unit length.
   $|\boldsymbol{a_i}| = |\boldsymbol{a_o}| = |\boldsymbol{n}| = 1$.
2. The relative rotation angle is expressed by direction vectors.
   $\boldsymbol{a_i} \times \boldsymbol{a_o} = \boldsymbol{n} \sin\theta, \boldsymbol{a_i} \cdot \boldsymbol{a_o} = -\cos\theta$.
3. The axis direction vector is defined by positions of both edges and link length.
   $\overrightarrow{P_i P_o} = L\boldsymbol{a_o}$.

Rotational velocity vectors of link axes

1. The relative rotational velocity is expressed by both axes' rotational velocity vectors and the rotational axis vector.
   $\boldsymbol{\omega_i} \cdot \boldsymbol{n} - \boldsymbol{\omega_o} \cdot \boldsymbol{n} = \omega$.

Transmission of rotaional velocity

1. The joint of link does not rotate around any axis normal to the rotational axis vector n.
   $\boldsymbol{\omega_o} \times \boldsymbol{n} = \boldsymbol{\omega_i} \times \boldsymbol{n}$.

Rigid body condition

1. The loading point is on the axis and the link is not bent.
   $\boldsymbol{v_e} - L_e\boldsymbol{\omega_o} \times \boldsymbol{a_o} = \boldsymbol{v_o} - L\boldsymbol{\omega_o} \times \boldsymbol{a_o} = \boldsymbol{v_i}$.

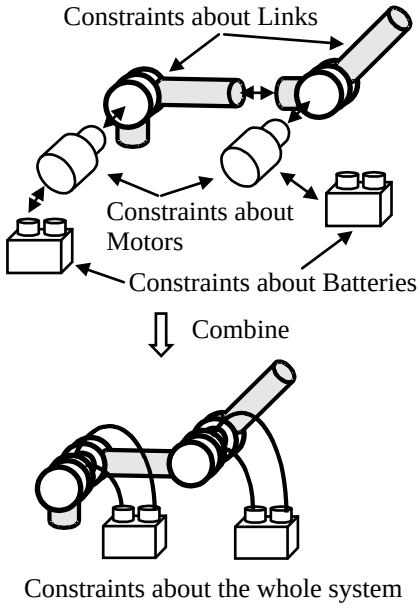


Figure 2. Composing the global constraint set

Force balance $\boldsymbol{F_i} + \boldsymbol{F_o} = \boldsymbol{F_e}$.
Torque balance $\boldsymbol{\tau_i} + \boldsymbol{\tau_o} - \boldsymbol{F_o} \times L\boldsymbol{a_o} = \boldsymbol{\tau_e} - \boldsymbol{F_e} \times L_e\boldsymbol{a_o}$.
Parameter list

$\boldsymbol{a_i}, \boldsymbol{a_o}$: axis direction vectors,
$\boldsymbol{n}$: rotational axis vector,
P_i: position of rotaional axis,
P_o: position of interface to an adjacent link,
L: link length,
Le: position of load,
θ: relative rotational angle between axes,
ω: relative rotational velocity between axes,
$\boldsymbol{\omega_i}, \boldsymbol{\omega_o}$: rotational velocity vectors of axes,
$\boldsymbol{v_i}, \boldsymbol{v_o}$: velocity vectors of edges,
$\boldsymbol{v_e}$: velocity vector at load point,
$\boldsymbol{\tau_i}, \boldsymbol{\tau_o}$: output torque,
$\boldsymbol{F_i}, \boldsymbol{F_o}$: output force,
$\boldsymbol{F_e}, \boldsymbol{\tau_e}$: external force and torque.

Figure 2 shows an example of composing a global constraint set about the product by defining interface conditions between basic components selected from the component library. In Figure 2, two links, two motors and two batteries are selected and combined, then a robot arm is constructed. A designer defines interface conditions including that interface points of both links are identical, that direction of link axses and rotation axses are same, and that output forces at the interface point counterbalance each other.

2.2 Algebraic constraint solver — Approach to computation of parameter values —

In order to determine parameter values, a designer has to analyse design solutions from different viewpoints such as forward and inverse kinematic analysis and optimization. To do that, it is necessary to produce various computer programs in accordance with the analyses, and this process can often be tedious and error-prone. In order to reduce designer's workload, the authors provide a designer with the

algebraic constraint solver [8] which automatically executes various analyses in response to designer's requests. A designer can obtain results of analyses only by giving the algebraic constraint solver the global constraint set about the design solution and designating the analysis. That is, the workload about making computer programs is reduced and a designer can concentrate on determining parameter values.

The algebraic constraint solver has to handle an ill-structured problem to support preliminary design. An ill-structured problem is defined as a problem in which there do not exist enough constraints to obtain solutions uniquely. At the early design stage, the number of constraints is often smaller than that of design parameters and there may be an infinite number of solutions. For example, suppose a constraint "a hand of the two-joint robot has to reach both point X and point Y, and each arm length has to be within a certain range" is given. Though it is described in terms of algebraic inequalities and equations, there are not enough equations to determine the concrete numerical value of each arm length and there may exist an infinite number of solutions. However, if the positions of point X and Y or the range of arm length are not appropriate, no solution can be found. In general, such an ill-structured problem is difficult to handle, and a designer is obliged to solve it in a trial and error manner based on designer's own knowledge and experience. As a result, it is often possible that a designer overlooks an existing solution or wastes time to search a solution which does not actually exist [8]. The algebraic constraint solver is developed to avoid these possibilities. It provides the following functions.

1. Finding out conflicts between inequalities
2. Detecting parameter values related to conflicts
3. Optimizing parameter values
4. Computing example parameter values
5. Plotting a graph of relationships between parameters

2.2.1 *Finding out conflicts between inequalities*

Let Eq. (1) be a constraint set about the design solution.

$$\begin{aligned} &f_1(\boldsymbol{x}) = 0, \ldots, f_p(\boldsymbol{x}) = 0, \\ &g_1(\boldsymbol{x}) \neq 0, \ldots, g_q(\boldsymbol{x}) \neq 0, \\ &h_1(\boldsymbol{x}) \geq 0, \ldots, h_r(\boldsymbol{x}) \geq 0, \end{aligned} \tag{1}$$

where $\boldsymbol{x} = (x_1, \ldots, x_n)$. Eq. (1) is replaced by Eq. (2) by introducing slack variables $\boldsymbol{s} = (s_1, \ldots, s_q)$ and $\boldsymbol{t} = (t_1, \ldots, t_r)$.

$$\begin{aligned} &f_1(\boldsymbol{x}) = 0, \ldots, f_p(\boldsymbol{x}) = 0, \\ &g_1(\boldsymbol{x}) \cdot s_1 = 1, \ldots, g_q(\boldsymbol{x}) \cdot s_q = 1, \\ &h_1(\boldsymbol{x}) = t_1, \ldots, h_r(\boldsymbol{x}) = t_r, \\ &t_1 \geq 0, \ldots, t_r \geq 0. \end{aligned} \tag{2}$$

The algebraic constraint solver generates Eq. (3) consisting of slack variables $t_1, \ldots, t_r$ from Eq. (2), and finds out conflicts according to the following method [8].

$$c(t_{i_1}, \ldots, t_{i_m}) = 0 \ (m \leq r). \tag{3}$$

Finding out conflicts
If all the coefficients of polynomial $c(t_{i_1}, \ldots, t_{i_m})$ and its constant term are positive, then it is impossible to simultaneously satisfy all the inequalities $h_{i_1}(\boldsymbol{x}) \geq 0, \ldots, h_{i_m}(\boldsymbol{x}) \geq 0$. That is, these m inequalities cause a conflict.

For example, if "$t_1 + 2t_2t_3 = -1$" is obtained, inequalities $h_1(\boldsymbol{x}) \geq 0, h_2(\boldsymbol{x}) \geq 0, h_3(\boldsymbol{x}) \geq 0$ cause a conflict and no solution exists. In order to remove such a conflict, a designer has two alternatives. One is to relax at least one inequality constraints, the other is to change some parameter values so that the conflict is resolved. The latter is discussed in the next section.

2.2.2 *Detecting parameter values related to conflicts*

When conflict is found, it is quite useful to suggest a method to remove the conflict. The algebraic constraint solver detects parameters whose values should be changed to remove conflict.

Suppose Eq. (3) shows the conflict. In general, a univariate linear equation (4) shows that value a is assigned to parameter x_k.

$$x_k = a. \tag{4}$$

Thus, if Eq. (4) is necessary to obtain Eq. (3), it is concluded that Eq. (4) has something to do with the conflict, that is, the conflict might be removed by changing the value of x_k. Otherwise, Eq. (4) has nothing to do with the conflict and the value of x_k needs not to be changed.

Let I be an ideal [2] of an equation set obtained by subtracting Eq. (4) from Eq. (2). If Eq. (3) does not belong to I, it is concluded that Eq. (4) is necessary to obtain Eq. (3). Otherwise, Eq. (3) can be obtained without Eq. (4), that is, Eq. (4) is unnecessary. Therefore, by using the Gröbner base [2] (see Appendix) of I, it is checked whether Eq. (4) has something to do with the conflict shown by Eq. (3) [8].

Relationship between Eq. (4) and the conflict shown by Eq. (3)
Let G be a Gröbner base of an equation set obtained by subtracting Eq. (4) from Eq. (2).

1. G does not reduce $c(t_{i_1}, \ldots, t_{i_m})$ to 0.
   $\Rightarrow$ Eq. (4) has something to do with the conflict shown by Eq. (3).
2. G reduces $c(t_{i_1}, \ldots, t_{i_r})$ to 0.
   $\Rightarrow$ Eq. (4) has nothing to do with the conflict shown by Eq. (3).

2.2.3 *Optimizing parameter values*

A designer often needs to conduct optimization such as "minimizing the weight" and "minimizing power consumption". The algebraic constraint solver minimizes or maximizes an objective function defined by the designer under the constraints of Eq. (1). In order to avoid computational divergence and rounding error, algebraic algorithm based on the Lagrange multiplier method is employed [8].

2.2.4 *Computing example parameter values*

Parameter values are not always determined based on rational decision. Rather, quit a few parameters are assigned numerical values considered to be appropriate based on designer's knowledge and experience. The algebraic constraint solver computes several example numerical solutions satisfying all the given constraints in order to give a designer a key to the optimum final design solution [8].

In order to compute example numerical solutions, the ill-structured problem is converted to a minimisation problem. First of all, an objective function $u(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{t})$ which has the following properties is introduced.

Objective function $u(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{t})$

1. $u(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{t})$ is continuous in $(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{t})$-space.
2. $u(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{t})$ has the minimum value in $(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{t})$-space.

3. If at least one parameter of $(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{t})$ becomes positive or negative infinite, $u(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{t})$ becomes positive infinite.

Let A be a region defined by Eq. (2). Since all the boundaries of A are included in A, $u(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{t})$ has the minimum value in A if A is not empty. Thus, by computing the minimum value of $u(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{t})$ in A, solutions to Eq. (2) are obtained as the minimum points. If $u(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{t})$ does not have the minimum value, it is concluded that A is empty, that is, Eq. (2) has no real solution.

The minimum value is computed by the optimizing function of the algebraic constraint solver.

2.2.5 Plotting a graph of relationships between parameters

Graphs showing relationships between two parameters are often used to determine parameter values and to check design solutions. The algebraic constraint solver computes an equation of two parameters x_i, x_j $(i < j)$ designated by a designer and plots the graph. Furthermore, it hatches the unacceptable area in which any point cannot be accepted as a solution.

Obtaining a plotting function

1. Gröbner base G_p of Eq. (2) is computed under the lexicographic ordering [2] in which x_i and x_j are given the lowest rank.
2. An equation consisting of x_i and x_j is selected from G_p. This is the plotting function.

Obtaining acceptable and unacceptable solution areas
An unacceptable area is given as a set of points which violate one of inequalities $t_1 \geq 0, \ldots, t_r \geq 0$ in Eq. (2). The acceptable and unacceptable solution areas are obtained according to the following procedure.

1. Gröbner base G_a of Eq. (2) is computed under the lexicographic ordering in which x_i, x_j and $t_1, \ldots, t_r$ are given the lowest rank.
2. Equations consisting of x_i, x_j and $t_1, \ldots, t_r$ are selected from G_a. By applying Quantifier Elimination technique [9] to the selected equations and $t_1 \geq 0, \ldots, t_r \geq 0$, the acceptable solution areas are computed. The unacceptable solution areas are obtained as the negation of the acceptable solution areas.

3 Architecture of the system

Figure 3 and Figure 4 show the architecture and an overview of the design support system respectively. The main part of the system is implemented in Visual C++, and the algebraic constraint solver is implemented in Risa/Asir [5] Windows edition.

A designer via the user interface works on a design problem. First of all, a product is decomposed into its sub-products in the *product manager*. In a product design, it is commonly agreed that a design problem can be decomposed into sub-problems. For each sub-product, design could be conducted separately.

The designer creates a product model by selecting appropriate basic components held in the *component library*, and editing the constraints via the *constraint editor*. In this system, each component is selected by drag & drop of the component icon. A line between component icons in the constraint editor shows that there exist interface conditions between those components.

All the created design partial solutions are classified and represented in the form of tree structure called *context-tree* [1] which represents the results of creating a full design solution from abstract to concrete. Each context-tree corresponds to one of sub-products defined in the product manager in one to one mapping. In a context-tree, all the generated constraints are inherited from the parent product design to the child one in an object-oriented manner.

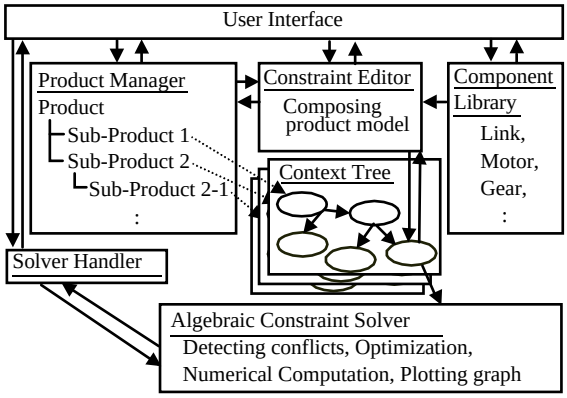


Figure 3. Architecture of the system

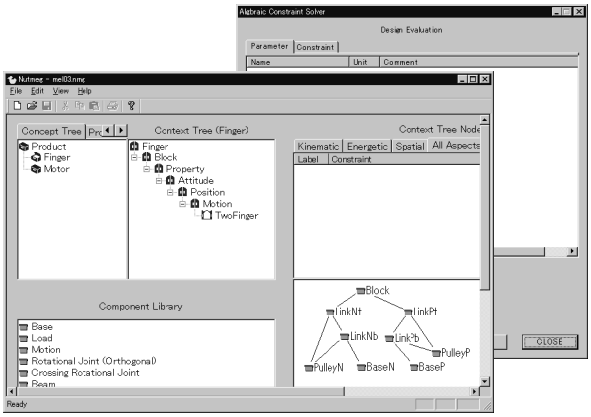


Figure 4. Overview of the system

When the designer gets a product design concrete to some extent, the *algebraic constraint solver* can be called via the *solver handler* and the product design solution is evaluated. This solver gives further information about a product design solution that helps a designer to progress towards an optimum final design solution. The solved results are shown to the designer through the solver handler.

4 A design example

This section describes an example of a design problem of wire-driven two-fingered-two-joint robot holding a cube (Figure 5). Suppose specifications and assumptions are given as below.

1. Specification

   Size of the cube 20 [mm] × 20 [mm] × 20 [mm]

   Mass of the cube 50 [g]

   Operation task To carry the cube from the point $(-60, 80)$ to $(60, 80)$ along to a horizontal line at the velocity of 100 [mm/sec]

2. Assumption

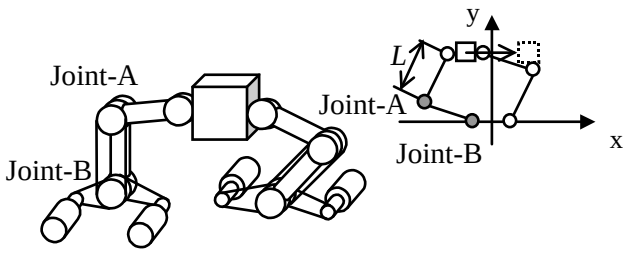


Figure 5. Two-fingered-two-joint robot

- Friction coefficient between the cube and robot's finger is 0.5.
- Length of the links are same.
- Link length L has the following relationship with its mass m.

$$m = \gamma L,$$

where γ is the linear density of link. Its value is assumed 3.5 [kg/m][3].

- The positions of robot's root joint are $x = \pm 20$ [mm].
- The left finger is designed. That is, the design task is *to determine the link length* L and *select appropriate motors and gear ratios for Joint-A and Joint-B* in Figure 5.

Design is conducted according to the following procedures.

1. Constructing the product model
2. Determining the fingertip force
3. *Determining the link length*
4. Computing the maximum power requirements at the joints
5. *Selecting the motors*
6. *Determining the appropriate gear ratios*
7. Analysing the obtained solution in detail

Details of each task are described below.

1. Constructing the product model
   Necessary components are selected from the component library and combined into a product model. The product model is expressed as a graph that shows interactions between components. If necessary, dimensions and interface conditions between components are defined. The system automatically generates a global constraint set about the product.
2. Determining the fingertip force
   A designer calls the algebraic constraint solver to plot a graph that shows relationship between cube's position and robot's fingertip force. In this example, though robot's finger tip force is not uniquely determined, its allowable range is shown in the form of inequalities as below.

$$\begin{aligned} F_x &\geq 490 \text{ [mN]}, \\ F_y &= 245 \text{ [mN]}, \end{aligned}$$

   where F_x and F_y are robot's finger tip force in x- and y-direction. Afterwards, F_x is supposed to be assigned 490 [mN].
3. *Computing the link length*
   In order to get an idea of possible link length, the algebraic constraint solver is called to plot a graph which shows relationship between cube's position and the link length. Figure 6 shows the graph drawn by the algebraic constraint solver. The vertical and horizontal axes show the link length [m] and cube's position [m] respectively. The hatched area shows the unacceptable area in which any point cannot be accepted as a solution. Mouse clicking on the graph invokes a dialog box that shows the detail information about the clicked point, the coordinate and inequality constraints' values. If the clicked point is in the hatched area, violated constraints are shown by marking them. This graph gives the information about possible link length to conduct the given operation. For example, if the link length is assigned 50 [mm] (black dotted line in Figure 6), the operation cannot be conducted within the marked region.
   Usually, it is preferable to assign a small value to link length and to make the total weight small. Figure 7 shows comparison of the necessary torque at each joint in the case between that the link length is 60 [mm] and that the link length is 80 [mm]. The vertical and horizontal axes represent the torque [Nm] and cube's position [m] respectively. In these graphs, points on black lines are valid and grey lines' points violate some constraints. Actually, in the case of shorter link length, the necessary torque is also smaller. These results of analysis give the designer grounds for design decisions. In this example, suppose the designer determines that the link length is 60 [mm].
4. Computing the maximum power requirements at the joints
   The designer calls the algebraic constraint solver to compute the maximum power requirements at the joints, and obtains the result of 34 [mW] for Joint-A and 212 [mW] for Joint-B.
5. *Selecting the motors*
   Based on the above maximum power requirements, the designer selects appropriate motors from commercial catalogues that can provide the necessary power. Suppose the following motor is selected for both joints.

Maximum power	250 [mW]
Rating voltage	3 [V]
Torque constant	1.93 [mNm/A]
Resistance	8.7 [Ω]

   These values can be added as new constraints about the motor.
6. *Determining the appropriate gear ratios*
   Usually, a gear ratio is firstly determined based on the maximum torque or the maximum rotational velocity, then evaluated and

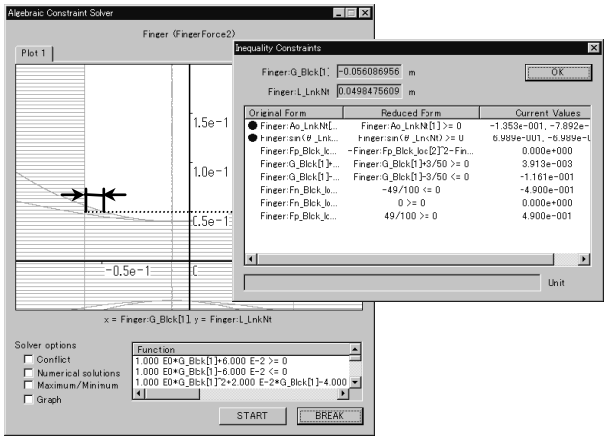
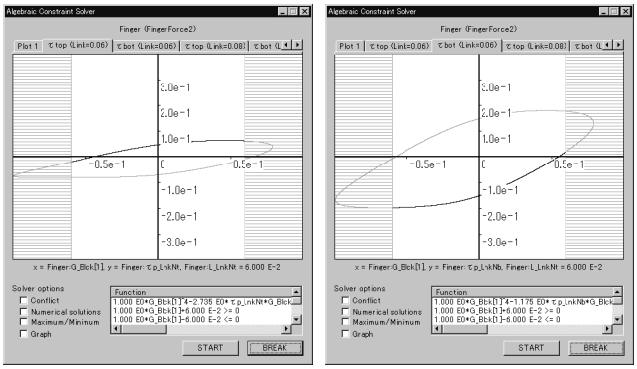


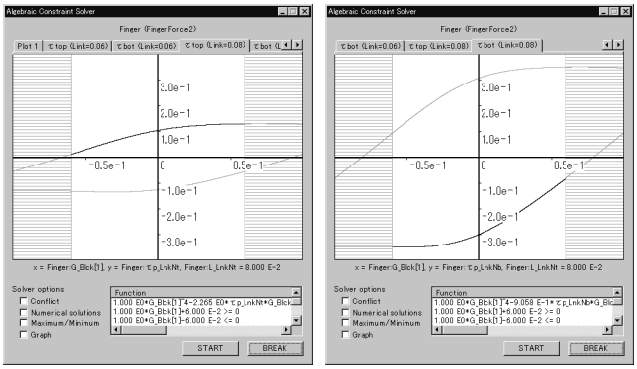
Figure 6. Link length to cube's position

[3] This value is obtained from a robot in [4].



Joint-A Joint-B

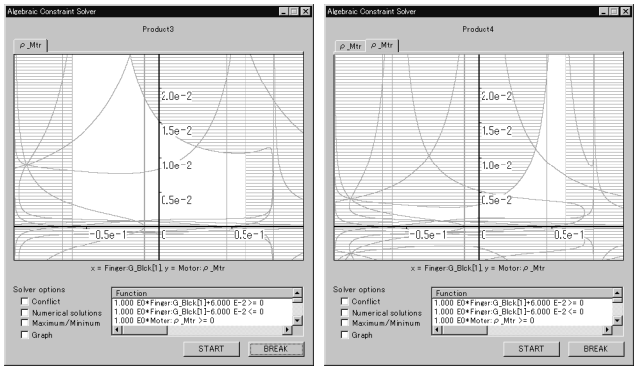
(a) Link length = 60 [mm]



Joint-A Joint-B

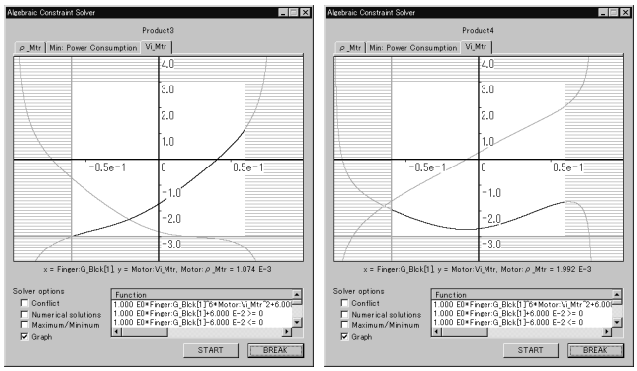
(b) Link length = 80 [mm]

Figure 7. Torque to cube's position



Joint-A Joint-B

Figure 8. Gear ratio to cube's position



Joint-A Joint-B

Figure 9. Input voltage to cube's position

modified so that the given operation can be conducted. However, in this example, in order to get an idea of possible gear ratios, the algebraic constraint solver is called to plot a graph between the gear ratio and cube's position. Figure 8 shows the graph where the vertical and horizontal axes represent the gear ratio and cube's position [m] respectively.

The designer can also call the algebraic constraint solver to get optimal gear ratios in the acceptable area shown in Figure 8. In this example, suppose the designer requires gear ratios which minimises the power consumption when the output power of each joint is maximum. Then, the algebraic constraint solver returns the gear ratio of 1/931 and 1/502 for Joint-A and Joint-B respectively.

7. Analysing the obtained solution in detail

   The designer can obtain further information about the robot by analysing the obtained solution in more detail. Such information gives the designer a good insight into the robot and its operation. For example, the designer can call the algebraic constraint solver to plot a graph that shows relationship between cube's position and input voltage to the motors.

   Figure 9 shows the graph in which the vertical and horizontal axes show the input voltage [V] and the cube's position [m] respectively. In Figure 9, points on black lines are valid solutions and grey lines' points violate some constraints. Violated constraints are shown in the dialog box invoked by mouse clicking on the grey lines. These graphs give the designer information about the appropriate input voltage to the motors to conduct the given operation.

This example contains more than 60 unknown design parameters and more than 70 constraints. All the computations were executed in the following environment.

OS	WindowsNT 4.0
CPU	Mobile Pentium-III 500MHz
RAM	320MB

Each computational time was from 1 second to 1 minute.

5 Conclusion

This research aims to help designers to gain an insight to a design problem, to reduce designer's workload and to improve working efficiency at the early design stage. These are to be achieved by developing a preliminary design support system. The system allows a designer to explore various design solutions and to examine them from different viewpoints efficiently. The system employs generic constraint solving techniques, especially computer algebraic techniques, and has the following advantages.

1. It allows a designer to compose and analyse various solution structures rapidly by combining basic components.
2. It provides computer tools for a designer to analyse design solutions and explore alternatives.

3. It automatically executes various analyses in response to designer's requests.

The above advantages are realised by applying computer algebraic techniques, which can handle generic constraints expressing design solutions. Especially, it is one of great advantages that a designer does not have to consider input and output parameters separately. For instance, in Section 4, the motor and the gear ratio were determined for the given robot structure. Instead, it is possible to determine the robot structure for the given motor in the same manner.

This system is currently developed for engineering design in the field of mechatronics. However, there is no limitation about the application area only if constraints can be expressed in the form of algebraic equations and inequalities. Therefore, it is possible to improve the system to deal with generic engineering design.

ACKNOWLEDGEMENTS

We would like to thank the Canon Foundation in Europe for their support for this research.

REFERENCES

[1] Akira Aiba and Ryuzo Hasegawa, `Constraint logic programming system – cal, gdcc and their constraint solvers –', in *Proceedings of International Conference on Fifth Generation Computer Systems*, pp. 113–131, (1992).

[2] David Cox, John Little, and Donal O'Shea, *Ideals, Varieties, and Algorithms*, Springer–Verlag, New York, 2 edn., 1996.

[3] M. J. French, *Conceptual Design for Engineers*, Springer–Verlag, 2 edn., 1985.

[4] Hitoshi Maekawa, Kazuhito Yokoi, Kazuo Tanie, Makoto Kaneko, Nobuo Kimura, and Nobuaki Imamura, `Development of a three-fingered robot hand with stiffness control capability', *Mechatronics*, **2**(5), 483–494, (1992).

[5] Masayuki Noro and Taku Takeshima, `Risa/asir – a computer algebra system', in *Proceedings of ISSAC'91*, pp. 387–396. ACM, (1992).

[6] Gerhard Pahl and Wolfgang Beitz, *Engineering Design*, Springer–Verlag, 2 edn., 1996.

[7] Stuart Pugh, *Total Design — Integrated Methods for Successful Product Engineering —*, Addison–Wesley Publishing Company, 1990.

[8] Hiroyuki Sawada, `The algebraic under constraint solver as a design tool', in *Proceedings of 1998 IMACS Conference on Applications of Computer Algebra (IMACS ACA'98)*, http:// math.unm.edu/ ACA/ 1998/ sessions/ industr/ sawada/ index.html, Prague, (1998).

[9] Volker Weispfenning, `Quantifier elimination for real algebra – the quadratic case and beyond', *Applicable Algebra in Engineering, Communication and Computing*, **8**, 85–101, (1997).

Appendix Brief Introduction of Gröbner Bases

When a finite set of multivariate polynomial equations is given, it can define an infinite set of polynomial equations called *ideal* [2]. Different equation sets defining the same ideal have the same solution set. Gröbner base is a simplified polynomial equation set that has the same solution set as the given equation set. This appendix gives some definitions of terminology and properties of Gröbner bases.

ideal Let $F = \{f_1(x_1, \ldots, x_n) = \ldots = f_k(x_1, \ldots, x_n) = 0\}$ be the given multivariate polynomial equation set. The ideal of F denoted by $Ideal(F)$ is defined as below.

$$Ideal(F) = \{h_1 f_1 + \cdots + h_k f_k = 0 | \\ h_j \ (1 \leq j \leq k) \text{ is a polynomial of } x_1, \ldots, x_n.\}.$$

Let $E = \{e_1(x_1, \ldots, x_n) = \ldots = e_m(x_1, \ldots, x_n) = 0\} \neq F$ be another polynomial equation set. If $Ideal(E) = Ideal(F)$, E has the same solution set as F.

Proof: Since $\forall(i)\{(e_i = 0) \in Ideal(E) = Ideal(F)\}$,
$\forall(i)\exists(h_1, \ldots, h_k)\{e_i = h_1 f_1 + \ldots + h_k f_k\}$.
Thus,
$$f_1 = \cdots = f_k = 0 \Rightarrow e_1 = \cdots = e_m = 0.$$
In the same way,
$$e_1 = \cdots = e_m = 0 \Rightarrow f_1 = \cdots = f_k = 0.$$
Therefore, E and F have the same solution set.

term A *term* is defined as a product of variables.

$$x_1^{\alpha_1} \ldots x_n^{\alpha_n} \ (\alpha_1, \ldots, \alpha_n \text{ are non-negative integers.}).$$

A monomial is a product of a coeffcient and a term, and a polynomial is a sum of monomials.

term order A *term order* $\prec$, which means "*less than*", is a linear order between terms that satisfies the following conditions.

1. $1 \prec p$ for all terms p.
2. $p \prec q \Rightarrow pr \prec qr$ for all terms p, q, r.

Example: Lexicographic ordering $(x_1 \prec \ldots \prec x_n)$

$$x_1^{\alpha_1} \ldots x_n^{\alpha_n} \prec x_1^{\beta_1} \ldots x_n^{\beta_n} \\ \Leftrightarrow \exists(i)\{\alpha_i < \beta_i \wedge \alpha_j = \beta_j (i+1 \leq j \leq n)\}.$$

head term A *head term* of a polynomial f is defined as the greatest term of f with respect to the term order $\prec$, and expressed as $ht(f)$.

reduce If a polynomial h has a term that is a multiple of a head term of a polynomial f, $f = 0$ *reduces* h by eliminating $ht(f)$ from h. Suppose h is reduced to a polynomial g, it is expressed as below.

$$h \rightarrow_f g.$$

Example: Suppose that $f = y - x^2 + x + 1$ and $h = z - xy + x^2$, and that the term order is lexicographic ordering $x \prec y \prec z$. Then, $f = 0$ reduces h by eliminating y as below.

$$h \rightarrow_f z - x(x^2 - x - 1) + x^2 = z - x^3 + 2x^2 + x.$$

normal form When a polynomial h is sequentially reduced by applying equations in a polynomial equation set F until it cannot be reduced anymore, this irreducible polynomial is called a *normal form* of h with respect to F and expressed as $h \downarrow_F$.

Gröbner base If a polynomial equation set G satisfies the following conditions, G is a Gröbner base of a polynomial equation set F.

1. $Ideal(G) = Ideal(F)$.
2. For any polynomial h, the normal form $h \downarrow_G$ is unique.

Gröbner bases are computed by Buchberger's algorithm [2].

Properties of Gröbner bases The followings are typical properties of Gröbner bases.

1. Let G be a Gröbner base. Then, $\forall(h \in Ideal(G))\{h \downarrow_G = 0\}$.
2. Let $F = \{f_1(x_1, \ldots, x_n) = \ldots = f_k(x_1, \ldots, x_n) = 0\}$ be a given polynomial equation set and G be a Gröbner base of F.
   $\forall(x_i)\exists((g = 0) \in G)\{ht(g)$ has a form of $x_i^{\alpha_i}.\}$
   $\Leftrightarrow F$ has a finite number of solution.
   This property is used to check whether the problem is ill-structured or not.
3. Let G be a Gröbner base under the lexicographic ordering $x_1 \prec \ldots \prec x_n$. G has the following form.
   $$G = \{g_1(x_1, \ldots, x_{n-1}, x_n) = g_2(x_1, \ldots, x_{n-1}) = \ldots \\ = g_{m-1}(x_1, \ldots, x_k, x_{k+1}) = g_m(x_1, \ldots, x_k) = 0\},$$
   where $m + k \geq n + 1$.
   This property is used to obtain plotting functions and solution areas.