Robotics: Science and Systems 2005
Cambridge, MA, USA, June 8-11, 2005

Optimal design of robots

J-P. Merlet
INRIA Sophia Antipolis
BP 93,06902 Sophia Antipolis,France
Email: Jean-Pierre.Merlet@sophia.inria.fr

***Abstract*— Synthesis of robots may be decomposed into two processes: *structural synthesis* (determine the general arrangement of the mechanical structure such as the type and number of joints and the way they will be connected) and *dimensional synthesis* (determine the length of the links, the axis and location of the joints, the necessary maximal joint forces/torques,...). The performances that may be obtained for a robot are drastically dependent on both synthesis. Although for serial robots general trends may be derived only from the structure a realistic comparison between two different structures may only be made after a careful dimensional synthesis and this is even more so for closed-loop robot (such as parallel robots).**

We will present a dimensional synthesis approach based on the design requirements that allows one to obtain almost all feasible design solutions that are guaranteed to satisfy the requirements, even taking into account manufacturing tolerances. Practical examples of 6-DOF robot design will be presented.

I. Introduction

Although robots are usually designed to perform a large variety of tasks it is not realistic to believe that a single robot will be flexible and performing enough to manage any task. On the other hand an end-user may wish to perform a a set of specific tasks with stringent requirements. Hence a fundamental question in robotics is to determine what is the most appropriate mechanical structure of the robot, being given tasks requirements (such as desired workspace, accuracy, load, stiffness, ...). Indeed it is not realistic to believe that sophisticated control algorithms coupled with a large number of sensors may be able to correct the behavior of a poorly designed robot. Furthermore on-board computer power should be more appropriately used for high-level tasks (such as planning, task management, interaction) instead of basic-levels control tasks that can be simplified by an appropriate mechanical design.

Design synthesis is a two-steps process:

- *structure synthesis*: determine the general arrangement of the mechanical structure such as the type and number of joints and the way they will be connected
- *dimensional synthesis*: determine the length of the links, the axis and location of the joints, ... In this paper the word dimension will have the broad sense of any parameter that will influence the robot behavior and is needed for the manufacturing of the robot

In some cases general trends for the robot performances may be deduced from the structure. For example we may compare the reachable workspace of serial 3 d.o.f. robot of type PPP and RRR: assuming a stroke of L for the linear actuator and a length L for the links the PPP workspace volume will be L^3 while it will be $\approx\ 40L^3$ for the RRR robot. Hence if a large workspace is required the RRR structure may seem to be more appropriate. But usually such trends will not be sufficient to fully determine the optimal robot: indeed many performances have to be taken into account for defining an optimal robot, some of them being highly dependent upon the dimensions of the robot (for example the load capacity). Furthermore such trends cannot be as easily derived for closed-chain robots.

Hence optimal design for a robot implies both type of synthesis. Our experience in the design of closed-chain robots has led us to the following rule of a thumb: *a robot with an a-priori more appropriate mechanical structure but whose dimensions have been poorly chosen will exhibit largely lower performances than a well dimensionally designed robot with an a-priori less appropriate structure*.

We are not claiming that structural synthesis is not an important area but that it cannot be disconnected from dimensional synthesis. The point is that structural synthesis, although still in progress, has strong theoretical backgrounds (such as screw and group theories) while, as we will see, dimensional synthesis lack of such background. Hence this paper will focus on dimensional synthesis.

II. Dimensional synthesis: state of the art

Dimensional synthesis is a problem that has attracted a lot of attention but most of the works focus on design for a specific robot's feature such as workspace [1], [5], [10], [13], [14] or accuracy [7], [21], [22] (this list is far from exhaustive and focus on closed-chain robots).

The usual way to solve the optimal design problem is to define a real-valued function C as a weighted sum of performance indices P_i [4]. These indices are real functions that define a "distance" between a requirement and the performance of a given robot with a value in the range [0,1]. A value equal to 0 indicates that the requirement is fully satisfied, a value larger than 0 indicates to which extent the requirement is violated while a value 1 is used when the requirement is fully violated. The performance indices are clearly functions of the design parameters set $\mathcal{P}$. The cost function is then defined as

$$C = \sum_i w_i P_i(\mathcal{P})$$

where w_i are weights. It is assumed that the optimal design solution is obtained for the value of the parameters in $\mathcal{P}$ that minimize C and a numerical procedure is used to find

the values of $\mathcal{P}$ which minimize C, usually starting with an initial guess $\mathcal{P}_0$ (note that already that the procedure used for the minimization should be able to find a global minimum of the cost function otherwise we may end up with only a local minimum).

But this method has many drawbacks. First it is assumed that the requirement indices can be defined, can be calculated efficiently (the numerical optimization procedure requires a large number of evaluation of these indices) and should be differentiable functions of the design parameters (otherwise finding the minimum of the cost-function may be quite difficult). All these assumptions are difficult to realize in practice for robots: for example what could be the definition of an index that indicates that a cube of given volume must be included in the robot's workspace ? Evaluation of some indices may also be a quite difficult problem: for example we may define as index the worst positioning error along a given axis for any pose of the robot within a prescribed workspace and evaluating this index is by itself a difficult constrained optimization problem. Furthermore index evaluation is complex as we look for *guaranteed* results (for example for the worst positioning error we want to be sure to have calculated the global maximum and not a local one). But calculating guaranteed results does not automatically imply that we need *exact* results. For example assume that we want to compare two different robots with respect to a given performance index and that we have an algorithm that provides a value V_a such that the real value V of the performance index satisfies $V \in V_a + [0, \epsilon_V]$ where ϵ_V is a user-defined upper bound for the algorithm error: although we will not compute the *exact* value of the performance index a guaranteed comparison between the 2 robots will be possible as soon as we are able to define a value for ϵ_V such that the ranges $V_a + [0, \epsilon_V]$ for the 2 robots have no intersection.

We will see later on that we may indeed design such algorithms and that interestingly their computation time is largely dependent upon ϵ_V. Such way to get a guaranteed result is considered as a strong alternative to calculating exact result that may be quite difficult to obtain because of complexity reasons or numerical round-off errors in the calculation.

Another drawback of the cost-function approach is the difficulty in the determination of the weights. These weights are present in the function not only to indicate the priority of the requirements but also to tackle with the units problem in the performance indices. For example for a 3-dof translational robot if the used performance indices are the workspace volume and positioning accuracy we are dealing with quantities whose units differ by a ratio of 10^3: hence the weights must be used to normalize the indices.

The choice of the weights is therefore essential while there is not intuitive rules for determining their values. Furthermore a small change in the weights may lead to very different optimal designs.

Even if the cost-function is effective it may lead to inconclusive result. This was exemplified by Stoughton [23] who was wanting to determine special kind of Gough platform with improved dexterity and a reasonable workspace volume. Hence Stoughton has considered two criteria in his cost-function: the dexterity and the workspace volume. He find out that these criteria were varying in opposite ways: the dexterity was decreasing when the workspace volume was increasing. Hence there was no optimal design solution per se and the problem was in fact to determine an acceptable compromise between the two requirements. This advocates the point that in optimal design we should not try to maximize one performance without imposing constraint on the minimal values of other performances (for example Gosselin [6] shows that the Gough platform having the largest workspace for a given stroke of the actuator will have a geometry such that it cannot be controlled). It may also be considered that in some cases some requirements are *imperative* i.e. they must never be violated while some others may be somewhat relaxed. But imposing an imperative requirements in the cost-function is difficult without violating the differentiability constraint and/or allowing large violation on the other constraints.

A final drawback of the cost-function approach is that it provides only one solution. This causes three main problems:

- *manufacturing tolerances* will be such that the real robot will differ from the theoretical one. Hence with only one theoretical design solution we cannot guarantee that the real robot will fulfills the requirements
- providing only one solution does not allow to consider secondary requirements that may have not been used in the cost-function but may be a decision factor if two robots satisfy in a similar way the main requirements
- for providing only one solution we have to assume that the designer masters all the criterion that will lead the end-user to a solution. This is seldom the case in practice: for example economic considerations will usually play a role although the designer cannot be fully aware of their level of implication

We will propose now another design methodology.

III. Another design methodology: the parameters space approach

We will first define the *parameters space* $\mathcal{S}^n$ as a n-dimensional space in which each dimension corresponds to one of the n design parameters of the robot. Hence a point in that space correspond to one unique design of the robot.

Consider now a list of m requirements $\{\mathcal{R}_1, \ldots, \mathcal{R}_m\}$ that define minimal or maximal allowed values of some robot's performance (such as accuracy, stiffness, ...) or some required properties (for example that a set of pre-defined trajectories lie within the robot's workspace) and assume that we are able for each requirement $\mathcal{R}_i$ to design an algorithm that is able to calculate the region $\mathcal{Z}_i$ defined as the region of the parameter space $\mathcal{S}^n$ that includes all the robot's design that satisfies the requirement $\mathcal{R}_i$. Then the intersection of all the $\mathcal{Z}_i$ defines **all** the robot's design that satisfies **all** the requirements. With this approach we will have found a **complete** answer to the optimal

design problems as we will have determined all possible design solutions.

To make this approach practical we are confronted to two difficulties:

1) calculating the region $\mathcal{Z}_i$
2) computing the intersection of the regions

The calculation of the region is indeed quite difficult as we have basically to determine regions whose borders are determined by a set of complex highly non-linear relations (but in some cases this may be possible if the number of design parameters is not too high, see [16], [19]). But a good point is that it is not necessary to determine these regions *exactly*. Indeed determining points of the region close to the border does not make sense as if they are chosen as nominal parameter value, then the real robot, whose parameter are affected by manufacturing tolerances, may in fact have a representative point in the parameters space that is outside the $\mathcal{Z}_i$ regions. Hence computing an *approximation* of the regions whose border is sufficiently close to the real border is sufficient.

The second point may also be difficult as computing the intersection of highly non-linear varieties may be quite difficult. To solve the intersection problem there are two non mutually exclusive approaches:

- describes (or approximates) the region by a set of geometrical objects whose intersection can be easily computed
- use the description of a region $\mathcal{Z}_i$ as an input for the calculation of the region $\mathcal{Z}_{i+1}$ i.e. determine only the points of $\mathcal{Z}_{i+1}$ that are also points of $\mathcal{Z}_i$. Using this approach there is no need to calculate the intersection of the regions as the output of the algorithm for region $\mathcal{Z}_i$ is already the intersection of the regions $\mathcal{Z}_1, \ldots, \mathcal{Z}_i$

The following parts of this paper describes preliminary algorithms that can be used for this design approach. These algorithms are based on interval analysis, a topics that we will now describe succinctly.

IV. Interval analysis

A. Interval arithmetics

Interval arithmetics [18] is a simple method that allows to determine lower and upper bounds for a function being given ranges for the unknowns appearing in the function. The *interval evaluation* of a function for given ranges for the unknowns is a method that allows to determine an interval that is guaranteed to include the exact lower and upper bounds of the function over the possible values of the unknowns in their ranges. Hence if $f(x_1, x_2, \ldots, x_n)$ is a function of the n unknowns x_i which are restricted to lie in the range X_i, then the interval evaluation of f gives two numbers A, B such that:

$$A \leq f(x_1, \ldots, x_n) \leq B \quad \forall\ x_i \in X_i, i \in [1, n]$$

The simplest interval evaluation method is the *natural evaluation* in which each mathematical operator $\diamond$ of the function is replaced by an interval equivalent $\diamond'$ returning an interval $[\underline{\diamond'}, \overline{\diamond'}]$ such that for all x in a range X $\underline{\diamond'(X)} \leq \diamond(x) \leq \overline{\diamond'(X)}$. Interval equivalent exist for all classical mathematical functions and hence interval arithmetics may be used in most cases: in particular all functions (algebraic, trigonometric, exponential) that occur in robotics can be evaluated with interval arithmetics.

Consider for example the function

$$f(x) = x^2 - 2x\ \sin(x) + \sin^2(x)$$

for x in [2,3]. The interval equivalent of the square function is defined by

$$[a, b]^2 = [0 \text{ if } 0 \in [a, b], \text{Min}(a^2, b^2) \text{ otherwise}, \text{Max}(a^2, b^2)]$$

Hence when x lie in the range [2,3], then x^2 lie in the range [4,16]. Using the property of the trigonometric function the interval evaluation of $\sin([2, 3])$ is approximately $[0.1411, 0.9092]$. Intervals may also be multiplied and added and finally the interval evaluation of f is approximately $[-1.4358, 9.2623]$.

Note that the interval evaluation of a function depends on the analytical form used to define the function. For example $f(x)$ may also be written as $(x - \sin(x))^2$ whose interval evaluation for x in [2,3] is [1.1896,8.1731]. Note that 0 is not included in this evaluation: this implies that f cannot cancel for x in [2,3].

As it may be noticed in the previous example the interval evaluation may not give the *exact* lower and upper bounds of the function (see the first interval evaluation): there may be an underestimation of the lower bound and an overestimation of the upper bound (but note that the second interval evaluation is exact for the given range: provided that we compute with an infinite accuracy the bounds of the evaluation are exactly the minimum and maximum of f).

A point is that the differences between the bounds of the interval evaluation and the exact minimum/maximum are strictly decreasing with the *width* of the range for the unknowns (i.e. the difference between the upper and lower bound of the ranges).

From a computer view point a very important property is that interval arithmetics may be implemented to take into account computer round-off errors. Any calculation using interval arithmetics is then guaranteed to includes the real value of the result. Computer errors are most often not taken into account in robotics but may play an important role. Consider the following example due to Rump: compute the value of $f(x, y)$:

$$333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + \frac{x}{2y}$$

for $x = 77617, y = 33096$. With Scilab or Matlab the computed value is about $-1\ 10^{23}$, in `C` we get 1.1726, the interval evaluation is $[-0.56610^{23}, 0.55510^{23}]$ while the real value is $\approx$ -0.8273960599. Hence even for a simple function computer errors may be quite large. Freely available packages implements interval arithmetics: for our tests we use the C++ package `BIAS/Profil`[1]

[1] http://www.ti3.tu-harburg.de/Software/PROFILEnglisch.html

B. Notation for interval analysis

The lower and upper bound of an interval X will be denoted $\underline{X}, \overline{X}$ and the width of this interval is $w(X) = \overline{X} - \underline{X}$. The midpoint of an interval X is defined as:

$$mid(X) = \frac{\overline{X} + \underline{X}}{2}$$

A n-dimensional interval set is called a *box*:

$$X = \{[\underline{X_1}, \overline{X_1}], \ldots, [\underline{X_n}, \overline{X_n}\} \quad (1)$$

The width of n-dimensional interval set X is the maximal width of its interval components.

Bisection is one of the most basic operation of interval analysis. For an n-dimensional interval set X the result of a bisection along the variable x_i is the two new interval set $L(X), R(X)$ defined by:

$$L(X) \triangleq \{[\underline{x}_1, \overline{x}_1], \cdots, [\underline{x}_i, (\underline{x}_i + \overline{x}_i)/2], \cdots, [\underline{x}_n, \overline{x}_n]\} \quad (2)$$

$$R(X) \triangleq \{[\underline{x}_1, \overline{x}_1], \cdots, [(\underline{x}_i + \overline{x}_i)/2, \overline{x}_i], \cdots, [\underline{x}_n, \overline{x}_n]\} \quad (3)$$

C. An application example of interval analysis

We will illustrate the principle of interval-based algorithms on a realistic application: the singularity detection for parallel robots (a detailed presentation of the algorithm can be found in [17]). Singularity is defined as the pose of the robot at which the determinant $|J^{-1}|$ of the inverse jacobian matrix of the robot vanishes. A practical consequence of coming close to a singularity is that the forces in the legs may become very large as these forces are obtained as a ratio whose denominator is $|J^{-1}|$. It is hence quite important to determine if a pre-defined workspace includes one or more singularities. As usually singularities must be avoided a designer may be interested in a fast algorithm that gives a straight `yes-no` answer about the presence of singularities in the prescribed workspace.

To design such an algorithm we note first that we may assume that we know a pose M_1 in the workspace (that is supposed to have a single component) and that we are able to compute the sign of the determinant at this pose (eventually using interval arithmetics to guarantee the sign): this hypothesis is not necessary but will simplify the presentation of the algorithm and we will assume, without lack of generality, that the sign is positive. The main point is that if we are able to find a pose (or a set of poses) M_2 in the workspace at which the determinant is negative, then any path connecting M_1, M_2 must cross a singularity and consequently at least one singularity exists in the workspace. On the other hand if we can prove that for any pose in the workspace the determinant is positive, then the workspace is singularity-free.

For the sake of simplicity we will assume that the workspace is defined as a set of ranges for the m parameters that define a pose of the robot. In term of interval analysis the workspace is a box $\mathcal{B}_0$.

In view of the above remark interval analysis seems an appropriate tool to solve this problem. Indeed as an analytical expression of the determinant is available we may compute an interval evaluation of the determinant for a given box for the pose parameters. If the lower bound of this interval evaluation is strictly positive then we are sure that for all poses in the box the determinant is positive and consequently that the box does not include a singularity. On the other hand if the upper bound of the interval evaluation is negative, then the determinant for any pose in the box will be negative: consequently any pose in this box is a pose of type M_2 and the workspace includes a singularity. The only case in which we cannot conclude is obtained when the lower bound is negative while the upper bound is positive. For this type of box we will proceed with a bisection that will produce 2 new boxes that will be stored for further processing.

Formally the algorithm uses a list of boxes $\mathcal{L}$ that initially has one element $\mathcal{B}_0$. The boxes in this list will be proceeded in sequence and during this process boxes may be added to the list. The k-th box in the list will be denoted by $\mathcal{B}_k$, the index k is used to denote which box is currently processed and n_k is the total number of boxes in the list when the algorithm starts processing box $\mathcal{B}_k$. We denote by J_k the interval evaluation of the determinant for the box $\mathcal{B}_k$, with lower bound $\underline{J_k}$ and upper bound $\overline{J_k}$. If at a given pose we cannot safely assert the sign of the determinant (because of computer round-off errors) a flag **F** will be raised. We start with $k = 0$ and the algorithm proceeds along the following steps:

1) if $k > n_k$ return **UNCERTAIN** if **F** has been raised, otherwise return **NO SINGULARITY**
2) if $\underline{J_k} > 0$ then $k = k + 1$, go to 1
3) if $\overline{J_k} < 0$ return **SINGULARITY**
4) if $\underline{J_k} \leq 0$ and $\overline{J_k} \geq 0$ and the width of $\mathcal{B}_k$ is 0, then raise **F**, $k = k + 1$, go to 1
5) bisect $\mathcal{B}_k$ at the variable having the range with the largest width, store the 2 resulting boxes at position n_k+1, n_k+2, $n_{k+1} = n_k + 2$, $k = k + 1$, go to 1

Although naive in term of efficiency (as we will see in the next section) the above algorithm is typical of interval analysis. Two main features are typical:

- the result is *guaranteed* if the result is **SINGULARITY** or **NO SINGULARITY**. If the algorithm returns **UNCERTAIN** this means that the current computer arithmetics does not allow to determine the sign of the determinant at some poses. In that case it is necessary to perform a local analysis with an extended arithmetics
- the algorithm is appropriate for a *distributed implementation*: as the processing of one box is independent from the processing of the other boxes we may use a master computer to manage the list $\mathcal{L}$ and to send boxes in this list to slave computers that perform a few iterations of the algorithm and returns the result to the master. If there is no singularity in the workspace the decrease in computation time compared to a sequential implementation is a little bit less than the number of slaves as there is a small overhead for the communication between the master and the slaves. On the other hand if a

singularity occurs, then the decrease in computation time may be larger than the number of slaves as a box with a negative determinant may be found early by a slave while it may have been processed quite late in the sequential version

- the algorithm may take into account the uncertainties in the modeling of the robot. Instead of using fixed values for the geometric parameters in the interval evaluation of the determinant we may use interval whose width will be the manufacturing tolerances. In practice this means that we are testing for singularities a *family* of robots that includes the real manufactured robot

The above algorithm has been implemented in a generic way for 6-dof robot: Maple is used to compute symbolically the determinant (which is the only part of the algorithm that is robot's dependent) and write the result in a file that is parsed for computing the interval evaluation. With the improvements proposed in the next section this algorithm run rather efficiently: in the worst observed practical cases we get the answer in less than 30 seconds on a 1.2GHz laptop.

The proposed algorithm may be extended in various ways. We may manage for example more complex workspace as soon as it can be enclosed in a bounding box and that we have a test to determine if a box is fully inside, fully outside or only partly inside the workspace. In that case the above algorithm is modified to discard any box that is fully outside the workspace or that is partly inside but for which the lower bound of the interval evaluation of the determinant is positive. Mechanical constraints on the passive joints of the robot may be incorporated by using the same principle.

D. Interval analysis is not a black box!

Basically the worst case complexity of interval-analysis based algorithms is exponential because of the use of the bisection process [12]. In the above algorithm this worst case complexity may be obtained if we have exactly one singular pose within the workspace but such case will very rarely occurs in practice.

But the naive implementation of the above algorithm will not be very efficient if some improvements are not added [8], [11]:

- *improvement of the interval evaluation*: the interval evaluation of the derivatives f_k of the determinant with respect to the pose parameter x_k may also be computed. If for one of these interval evaluation the lower bound is strictly positive or the upper bound is strictly negative, then the determinant is monotonic with respect to the variable in the box. We may hence substitute the interval value of the variable by the lower or upper bound of its range to compute the lower and upper bounds of the determinant which is the purpose of an interval evaluation. This calculation must be done recursively: indeed assume that the interval evaluation of the derivatives f_l for l from 1 to $i-1$ has led to intervals with negative lower bound and positive upper bound while the interval for the derivative f_i has a positive lower bound. To compute the lower bound of the determinant we will use the value $\underline{x_i}$ for x_i instead of the range $X_i = [\underline{x_i}, \overline{x_i}]$. But during the calculation of the derivatives f_l with l up to $i-1$ we have used X_i as value for x_i: now that x_i has a fixed value the interval for some derivatives may change to have a constant sign, thereby allowing to fix another variable
- *filtering*: some heuristics allows one to decrease the width of a box "in place", i.e. without using bisection [2]. Consider for example that we must determine what are the values of x, y such that $x + y \leq 2$ when x lie in [0,4] and y in [-1,1]. We may write the above inequality as $x \leq 2 - y$. If we compute the interval evaluation of the right term we get [1,3] which implies that $x \leq 3$: the range for x may thus be substituted by [0,3]

Numerous other methods, some with parameters, may be used to improve the computation time of interval-based algorithms and thus a high level of expertise may be needed to make the algorithm works in practice. To conclude memory storage is often mentioned as a limitation of interval analysis but in our experience a careful storage management allows one to solve most problem with a number of storage boxes that has not to exceed 100.

V. Optimal design

We have seen that our optimal design approach requires the calculation of the regions $\mathcal{Z}$ and then their intersection. Interval analysis seems to be quite appropriate for the second part. Indeed if we assume that we are able to obtain the regions $\mathcal{Z}$ as a set of boxes, then calculating their intersection is a classical problem in computational geometry that can be solved easily.

We are now confronted to the problem of calculating the region $\mathcal{Z}$ using interval analysis. As mentioned previously there is no need to calculate *exactly* these regions as points on the border cannot be considered as nominal design parameter values because the effect of manufacturing tolerances may put the value of the *real* robot parameter outside the region $\mathcal{Z}$. This point may be used as an advantage for interval analysis-based method by using the following rule:

the result of the algorithm should be a set of boxes such that for each box the range for each design parameter has a width which is at least equal to the manufacturing tolerance for this parameter

The rational behind this rule may be illustrated on an example. Assume that for a given parameter whose manufacturing tolerance is $[-\epsilon, \epsilon]$ the algorithm provides the result range $[a, b]$. If $b - a \geq 2\epsilon$ then we may choose as nominal value for the parameter any value in the range $[a + \epsilon, b - \epsilon]$: indeed to any such value we may add an arbitrary manufacturing tolerance in the range $[-\epsilon, \epsilon]$ with a result still in $[a, b]$. In other words the parameter value for the *real* robot will still be such that its representative points in the parameters space will belong to $\mathcal{Z}$.

Interval analysis-based method may be thought as a method to compute an *approximation* of the region $\mathcal{Z}$ in which the parts of $\mathcal{Z}$ that are too close to the border are eliminated. We will comment later on on this statement.

We have now to explain how we may design an algorithm to calculate the region $\mathcal{Z}$. For that purpose we will illustrate the principle on the singularity detection problem.

A. Calculating $\mathcal{Z}$: the singularity example

Consider now that the inverse jacobian matrix is a function not only of the pose parameters $\mathbf{X}$ but also of a set of m design parameters $\mathcal{P} = \{P_1, \ldots, P_m\}$ that are constrained to lie in some ranges: hence the set of design parameters must be included in a box $\mathcal{P}_0$. Each parameters P_i has a manufacturing tolerance $[-\epsilon_i/2, \epsilon_i/2]$. The problem is now to find possible values for the design parameters such that the corresponding robots are singularity-free over the workspace $\mathcal{W}_0$.

The algorithm described in section IV-C, denoted $\mathcal{A}_1$, will be used with two modifications:

- only a limited number N of bisection will be allowed and the algorithm will return **FAIL** if this number is exceeded.
- the value of the design parameters are now intervals. A direct consequence is that at a given pose the determinant may not have a constant sign: hence it may be difficult to find a pose M_1 at which the sign is constant. But the algorithm may start without this knowledge and attributes a sign for the determinant as soon as it finds a box in the list of $\mathcal{A}_1$ for which the determinant has a constant sign.

The algorithm uses a list of boxes $\mathcal{L}_P$ that initially has one element $\mathcal{P}_0$. The k-th box in the list will be denoted by $\mathcal{P}_k$, the index k is used to denote which box is currently processed and n_k is the total number of boxes in the list when the algorithm starts processing box $\mathcal{P}_k$. We will denote $\mathcal{A}_1(\mathcal{P}_k)$ a call to the algorithm $\mathcal{A}_1$ when the design parameters have as possible values the range described in the box $\mathcal{P}_k$. The output of the algorithm is a file, called the *result file* that describes all the parameters boxes defining the geometries of singularity-free robots. We start with $k = 0$ and the algorithm proceeds along the following steps:

1) if $k > n_k$ **EXIT**
2) if $\mathcal{A}_1(\mathcal{P}_k)$=**SINGULARITY** then $k = k + 1$, go to 1
3) if $\mathcal{A}_1(\mathcal{P}_k)$=**NO SINGULARITY** then store $\mathcal{P}_k$ in the result file, $k = k + 1$, go to 1
4) if $\mathcal{A}_1(\mathcal{P}_k)$=**FAIL** or **UNCERTAIN** then
   a) if $w(P_j) < \epsilon_j$ for all j in [1,m], then $k = k + 1$, go to 1
   b) bisect $\mathcal{P}_k$ at the variable l having the range with the largest width and verifying $w(P_l) \geq 2\epsilon_l$
   c) store the 2 resulting boxes at position n_k+1, n_k+2, $n_{k+1} = n_k + 2$, $k = k + 1$, go to 1

The efficiency of this algorithm is influenced by the complexity of the determinant formulation but also by the parameter N. In general for boxes $\mathcal{P}_k$ having a large width it is useless to have a large N. On the opposite N may be large as soon as the width of the box come close to the manufacturing tolerances. Hence the value of N should be an increasing function of the boxes width that is usually empirically determined.

B. Calculating $\mathcal{Z}$: other examples

Apart from the singularity detection algorithm we have implemented another algorithm that deals with workspace constraints [9]. This algorithm allows one to determine the design parameters such that a given workspace (that may be specified as a set of poses, of segments in the 3D space or as a set of 6D boxes) must be included in the workspace of the robot.

Up to now we have assumed that the performance requirement has a closed-form that can be interval evaluated. This is not always the case in robotics. For example assume that we consider the positioning accuracy $\mathbf{\Delta X}$ of the robot with respect to the joint measurement errors $\mathbf{\Delta\Theta}$. Both quantities are linearly related by

$$\mathbf{\Delta X} = J(\mathbf{X})\mathbf{\Delta\Theta}$$

where J is the Jacobian matrix of the robot, whose elements are functions of the pose $\mathbf{X}$ and of the design parameters.

The following requirement is classical: being given bounds $\mathbf{\Delta\Theta}^M$ on the joint errors determine the design parameters such that the robot's positioning errors are lower than given thresholds $\mathbf{\Delta X}^M$, whatever is the pose of the robot in a given workspace $\mathcal{W}$. Unfortunately for closed-chain robots the matrix J may be quite complex (or even may not be available) while its inverse J^{-1} may have a simple form. But it is possible to state the problem using only J^{-1}: find the design parameters $\mathcal{P}$ such that for all $\mathbf{X}$ in $\mathcal{W}$ all the solutions in $\mathbf{\Delta X}$ of the linear system $J^{-1}(\mathbf{X}, \mathcal{P})\mathbf{\Delta X} = \mathbf{\Delta\Theta}$ with $\mathbf{\Delta\Theta} \leq \mathbf{\Delta\Theta}^M$ are included in $\mathbf{\Delta X}^M$.

We have thus to solve a classical problem of interval analysis: being given an interval matrix $\mathbf{A}$ and an interval vector $\mathbf{b}$ determine an enclosure of all the solutions of the linear interval system $\mathbf{A}x = \mathbf{b}$ i.e. a region that includes the solution of $Ax = b$ for all A, b included in $\mathbf{A}, \mathbf{b}$ [18], [20]. It can be shown that classical methods of linear algebra (such as the Gauss elimination algorithm) may be extended to deal with this problem. We may directly use these methods to compute an enclosure of $\mathbf{\Delta X}$ and store as result the parameters boxes such that this enclosure is included in $\mathbf{\Delta X}^M$. But we may improve their efficiency: indeed these methods assume no dependency between the elements of $\mathbf{A}$ i.e. the elements of the matrices A that are considered may have any arbitrary value within their ranges in $\mathbf{A}$. In our case there are dependencies between the elements of J^{-1} and not all possible values are allowed.

Our basic method is the Gauss elimination scheme. We compute an interval evaluation $\mathbf{A}^{(0)}$ of $\mathbf{A}$ and an interval evaluation $b^{(0)}$ of $\mathbf{b}$ (using the derivatives of the components of $\mathbf{A}, \mathbf{b}$ to improve these interval evaluations). The Gauss

elimination scheme may be written as [20]

$$A_{ik}^{(j)} = A_{ik}^{(j-1)} - \frac{A_{ij}^{(j-1)} A_{jk}^{(j-1)}}{A_{jj}^{(j-1)}} \quad \forall\, i \text{ with } j > k \quad (4)$$

$$b_i^{(j)} = b_i^{(j-1)} - \frac{A_{ij}^{(j-1)} b_j^{(j-1)}}{A_{jj}^{(j-1)}} \quad (5)$$

The enclosure of the variable X_j can then be obtained from $X_{j+1}, \ldots, X_n$ by

$$X_j = (b_j^{(j-1)} - \sum_{k>j} A_{jk}^{(j-1)} X_k)/A_{jj}^{(j-1)} \quad (6)$$

We have improved the interval evaluation of the quantities appearing in the scheme by taking into account the derivatives of the elements of $A^{(0)}, b^{(0)}$ with respect to the pose and design parameters and propagating them by using the derivatives of the elements of $A^{(j-1)}$ to calculate the derivatives of the elements of $A^{(j)}$ and use them for the interval evaluation. Our experiments have shown that this lead to a drastic increase in term of the tightness of the enclosure.

Note also that this method may be used to determine what should be the design parameters so that any wrench in a set may be produced at any pose of $\mathcal{W}$ while the joint forces/torques are bounded. By duality the method can also solve the velocity problems (for bounded joint velocities find the design parameters such that any end-effector twist in a given set may be realized at any pose in $\mathcal{W}$).

C. A critical analysis of the zone calculation

We have presented in the previous section various methods to compute an approximation of the region $\mathcal{Z}$. However it is not possible to claim that we guarantee to get an approximation of the region that includes all possible values of the design parameters, up to the manufacturing tolerances, that will satisfy the performance index. Indeed for complex performances index the overestimation of interval arithmetics may be so large that only for very small boxes (i.e. whose width is lower than the manufacturing tolerances) we can guarantee that the performance index is satisfied. But the union of such small boxes, that may exist in the intersection of the $\mathcal{Z}_i$, may constitute boxes whose final width may be larger than the manufacturing tolerances.

Our experience however is that for robotics problem this is not the case. But a possibility to tackle this problem is to assume that the tolerances are much lower than then real one. After calculating the approximation of the regions and their intersection we may then decrease the result by the real tolerances to get a safe design region.

D. Calculating the intersection of the $\mathcal{Z}$

As soon as an approximation of the regions $\mathcal{Z}_i$ have been determined as a set of boxes in the parameters space calculating their intersection is a classical problem of computational geometry with complexity $O(n \log n)$ for n boxes. But calculating the intersection may be avoided in a way that even speed-up the total calculation. Indeed assume that the region $\mathcal{Z}_1$ has been computed for the first requirement, leading to a list of boxes $\mathcal{L}^1$. For the second requirement instead of using $\mathcal{P}_0$ as single element of the list $\mathcal{L}_P$ (and thus looking for all parameters that satisfy the second requirement) we may use $\mathcal{L}^1$ as $\mathcal{L}_P$, thereby looking only for the parameters that satisfies both requirements. Proceeding along this line for all requirements will lead to a result that satisfy all requirements. A drawback however is that if one of the algorithm fail to provide design solutions (or if we want to modify a requirement) we may have to restart a large part of the calculation.

E. The algorithm in practice

As mentioned previously the algorithm are implemented in C++ using `BIAS/Profil` for interval arithmetics and our own interval analysis library `ALIAS`[2] that offer high-level modules that are combined for implementing the calculation of the region $\mathcal{Z}$.

F. Choosing the optimal design

Assume now that we have succeeded in computing the regions for all requirements and then their intersection $\mathcal{Z}_\cap = \cap \mathcal{Z}_i$. Clearly we cannot propose to the end-user an infinite set of solutions and our purpose is now to propose various design solutions whose representative points lie in $\mathcal{Z}_\cap$ (i.e. they satisfy the requirements). But a robot presents various performances, denoted secondary requirements, that may not be part of the main requirements but which can be used to help choosing the best design. Ideally the presented design solutions should be representative of various compromises between the secondary requirements. Unfortunately there is no known method to deal with this problem. Hence we just sample the region $\mathcal{Z}_\cap$ using a regular grid, compute the secondary requirements at the nodes of the grids and retain the most representative solutions.

Note that the algorithms for computing the region $\mathcal{Z}$ may also be used to verify that a given design (or a small family of design as, for example, the family of robots whose design parameters have values around nominal values and within manufacturing tolerances, called the *family of manufactured robot*) satisfy a requirement, in which case they will be much more faster. Using this property and as we will provide finally only a finite set of design solution we may relax the requirements when computing the regions. For example for the workspace algorithm instead of specifying a whole 6D region as desired workspace we may specify only a finite set of poses: this will allow a faster calculation of the region in the parameters space and we will only have to verify that the proposed final design solution indeed includes the whole 6 workspace.

Similarly it may happen that for a specific requirement an algorithm for computing the region $\mathcal{Z}$ is not available. But as soon as an algorithm for verifying the requirement for the family of manufactured robots is available our design method may still be applied.

[2] www.inria-sop.fr/coprin/logiciel/ALIAS/ALIAS.html

G. Applications

As mentioned previously we have developed algorithms for computing the region $\mathcal{Z}$ for the following requirements: workspace, singularity detection, accuracy, velocity and static analysis. Such requirements are the most frequently encountered for practical applications. The design methodology has then been used for various practical applications: design of our own prototypes (for example the micro-robot MIPS for medical application [15]), fine positioning devices for the European Synchrotron Radiation Facility (ESRF) with a load over one tons and an absolute accuracy better than a micrometer [3], the CMW milling machine for high-speed manufacturing [24]. We are currently using this design methodology approach with Alcatel Space Industry for the development of an innovative deployable space telescope.

In each of these cases the on-the-shelf algorithms for calculating the region $\mathcal{Z}$ has to be adapted to deal with specificities of the application (for example the large workspace for the CMW milling machine implies that we have to deal with passive joint limits while the ESRF one, with a reduced workspace, such limits do not play a role). But the flexibility of interval analysis is large and has allowed us to solve the problem.

VI. Conclusion

The proposed design methodology has the main advantages of providing a large panel of design solution with a guarantee on the satisfaction of the main requirements, even taking into account manufacturing tolerances. However its practical implementation needs some expertise in interval analysis for the algorithm to be efficient. A current restriction is that only non time-dependent requirements (i.e. requirements that are not solution of a differential equations) may be taken into account: for example we cannot deal with dynamics. However there is no theoretical impossibilities to deal with these requirements with interval analysis and this is a prospective for our work.

The development of this methodology has been guided by applications in very different domains: manufacturing, fine positioning, space and medical applications.

Finally the methodology has been developed to deal with robots and mechanisms design but may be extended to problems in other area as well.

References

[1] Boudreau R. and Gosselin C.M. The synthesis of planar parallel manipulators with a genetic algorithm. *ASME J. of Mechanical Design*, 121(4):533–537, December 1999.

[2] Collavizza M., F. Deloble, and Rueher M. Comparing partial consistencies. *Reliable Computing*, 5:1–16, 1999.

[3] Comin F. Six degree-of-freedom scanning supports and manipulators based on parallel robots. *Review of Scientific Instruments*, 66(2):1665–1667, February 1995.

[4] Erdman A.G. *Modern Kinematics*. Wiley, New-York, 1993.

[5] Faraz A. and Payandeh S. A robotic case study: optimal design for laparoscopic positionning stands. In *IEEE Int. Conf. on Robotics and Automation*, pages 1553–1560, Albuquerque, April, 21-28, 1997.

[6] Gosselin C. *Kinematic analysis optimization and programming of parallel robotic manipulators*. Ph.D. Thesis, McGill University, Montréal, June, 15, 1988.

[7] Han C-S, Tesar D., and Traver A. The optimum design of a 6 dof fully parallel micromanipulator for enhanced robot accuracy. In *ASME Design Automation Conf.*, pages 357–363, Montréal, September, 17-20, 1989.

[8] Hansen E. *Global optimization using interval analysis*. Marcel Dekker, 1992.

[9] Hao F. and Merlet J-P. Multi-criteria optimal design of parallel manipulators based on interval analysis. *Mechanism and Machine Theory*, 40(2):151–171, February 2005.

[10] Huang T., Jiang B., and Whitehouse D.J. Determination of the carriage stroke of 6-PSS parallel manipulators having the specific orientation capability in a prescribed workspace. In *IEEE Int. Conf. on Robotics and Automation*, pages 2382–2385, San Francisco, April, 24-28, 2000.

[11] Jaulin L., Kieffer M., Didrit O., and Walter E. *Applied Interval Analysis*. Springer-Verlag, 2001.

[12] Kreinovich V., Lakeyev A., Rohn J., and Kahl P. *Computational complexity and feasibility of data processing and interval computations*. Kluwer, 1998.

[13] Lee E., Mavroidis C., and Merlet J-P. Five precision points synthesis of spatial RRR manipulators using interval analysis. *ASME J. of Mechanical Design*, 126(5):842–849, September 2004.

[14] McCarthy J.M. Mechanism synthesis theory and the design of robots. In *IEEE Int. Conf. on Robotics and Automation*, pages 55–60, San Francisco, April, 24-28, 2000.

[15] Merlet J-P. Optimal design for the micro robot MIPS. In *IEEE Int. Conf. on Robotics and Automation*, Washington, May, 11-15, 2002.

[16] Merlet J-P. Designing a parallel manipulator for a specific workspace. *Int. J. of Robotics Research*, 16(4):545–556, August 1997.

[17] Merlet J-P. and Daney D. A formal-numerical approach to determine the presence of singularity within the workspace of a parallel robot. In F.C. Park C.C. Iurascu, editor, *Computational Kinematics*, pages 167–176. EJCK, Seoul, May, 20-22, 2001.

[18] Moore R.E. *Methods and Applications of Interval Analysis*. SIAM Studies in Applied Mathematics, 1979.

[19] Murray A.P., Pierrot F., Dauchez P., and McCarthy J.M. A planar quaternion approach to the kinematic synthesis of a parallel manipulator. *Robotica*, 15(4):361–365, July - August , 1997.

[20] Neumaier A. *Interval methods for systems of equations*. Cambridge University Press, 1990.

[21] Pittens K.H. and Podhorodeski R.P. A family of Stewart platforms with optimal dexterity. *J. of Robotic Systems*, 10(4):463–479, June 1993.

[22] Ryu J. and Cha J. Volumetric error analysis and architecture optimization for accuracy of HexaSlide type parallel manipulators. *Mechanism and Machine Theory*, 38(3):227–240, March 2003.

[23] Stoughton R. and Arai T. A modified Stewart platform manipulator with improved dexterity. *IEEE Trans. on Robotics and Automation*, 9(2):166–173, April 1993.

[24] Wildenberg F. Calibration for hexapod CMW. In *2nd Chemnitzer Parallelkinematik Seminar*, pages 101–112, Chemnitz, April, 12-13, 2000.