Dimensional Synthesis of Parallel Robots with a Guaranteed Given Accuracy over a Specific Workspace

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

D. Daney
INRIA
BP 93, 06902 Sophia-Antipolis Cedex, France
David.Daney@sophia.inria.fr

***Abstract*— We are considering a n d.o.f. parallel robot that has to move within a given workspace and whose geometry is defined by a set of parameters. The motion of active joints of the manipulator are measured with sensors with a known accuracy $\pm\Delta\rho$. These errors together with bounded manufacturing errors on the parameters describing the geometry of the robot induces a positioning errors $\Delta\mathbf{X}$ of the platform. We present an algorithm that allows one to determine geometries of the robot ensuring that these positioning errors will lie within pre-specified limits for any pose of the robot in its workspace even if the physical realization of the robot differs from the theoretical model while staying within the given manufacturing errors bounds. A by-product variant of this algorithm allows one to compute the maximal positioning errors of a given robot up to a pre-defined accuracy.**

I. Introduction

It is well known that the performances of parallel robots are very sensitive to their geometry. In this paper we will consider a Gough platform (but the approach may be extended to any type of parallel robot as soon as its inverse jacobian has an analytic form) and we intend to determine the geometries of this type of robot such that the errors in the positioning of the platform lie within prescribed intervals.

Error analysis is a complex problem that has been mostly addressed in term of finding the positioning errors of a given robot at some specific location within the workspace [1], [2], [4], [3], [7], [8], [12], [13], [15], [16], [17], [18]. Our approach differs first because we will consider not only specific poses but the whole workspace. Then we will mostly consider the *error synthesis* problem i.e. finding the geometries of the robot that lead to have at least a required accuracy although *error analysis* will be possible by using a variant of our algorithm. Finally we will also take into account the differences between the theoretical geometrical model of the robot and the real values of this model so that any real robot manufactured according to our design solutions will have the required accuracy.

It is well known that the positioning errors $\Delta\mathbf{X}$ of the platform are related to the leg lengths measurement errors $\Delta\rho$ by:

$$\Delta\rho = J^{-1}(\mathcal{P}, \mathbf{X})\Delta\mathbf{X} \tag{1}$$

where J^{-1} is the inverse jacobian matrix of the robot, that is pose dependent but also depends on the geometrical parameters $\mathcal{P}$ that defines the geometry of the robot.

This geometry is defined by the location of the attachment points A_i (B_i) of the legs on the base (platform). We define a reference frame $O, \mathbf{x}, \mathbf{y}, \mathbf{z}$ and a mobile frame $C, \mathbf{x_r}, \mathbf{y_r}, \mathbf{z_r}$. We will not assume that the A_i or the B_j are coplanar but we will assume that the projection of the A_i's on a plane that is perpendicular to $\mathbf{z}$ are located on a circle with radius R_1 while the projection of the B_j's on a plane perpendicular to $\mathbf{z_r}$ are located on a circle of radius r_1 (figure 1). We define the angle θ_i, α_i in such way that

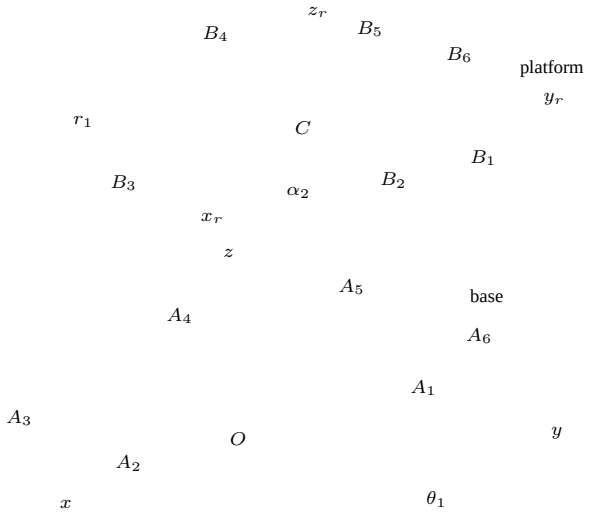

Fig. 1. The geometrical parameters of the robot

the coordinates of the A_i, B_j in the reference and mobile frames are:

$$\begin{aligned} A_i &= (R_1\cos\theta_i, R_1\sin\theta_i, z_i) \\ B_j &= (r_1\cos\alpha_j, r_1\sin\alpha_j, zb_j) \end{aligned}$$

Hence we have in $\mathcal{P}$ a total of 26 geometrical parameters: R_1, r_1 and 6 quadruples $\theta_i, z_i, \alpha_i, zb_i$. We will assume that all these parameters are bounded i.e. we are looking only for values of these parameters that lie within a given set of ranges $\mathcal{R}_P$.

The pose $\mathbf{X}$ of the robot will be defined by the coordinates x_c, y_c, z_c of the center of the platform C in the reference frame together with 3 angles ϕ_1, ϕ_2, ϕ_3 that allows one to calculate the rotation matrix R between the mobile and reference frame.

We assume that the robot will have to move within a given workspace $\mathcal{W}$ that is supposed to be defined as intervals for the $x_c, y_c, z_c, \phi_1, \phi_2, \phi_3$ parameters (but the approach may be extended to more complex workspace geometries as we will see later on, see section III-C).

We have a desired vector of maximal positioning errors $\Delta\mathbf{X}_d$ that is defined as a set of allowed ranges for the errors on x_c, y_c, z_c and for the angular errors. We will denote by $\Delta\mathbf{X}_d^i$ the i-th component of this vector.

Equation (1) may be rewritten as:

$$\Delta\mathbf{X} = J(\mathcal{P}, \mathbf{X})\Delta\rho \tag{2}$$

The element at i-th row and j-th column of J will be denoted as J_{ij} and we define the norm $|J_i|$ of the i-th row of the jacobian as

$$|J_i| = \sum_{k=1}^{k=6} |J_{ik}|$$

Note that as (2) is a linear system we may choose $\pm$ 1 as value for the sensor error and scale $\Delta\mathbf{X}_d$ correspondingly. Under this assumption the maximal absolute value of $\Delta\mathbf{X}_i$ is exactly $|J_i|$.

Our first goal is to find robot geometries for which we can ensure that whatever is the pose of the robot within the workspace the positioning error will be included in $\Delta\mathbf{X}_d$. But our second goal is that this feature will still hold even if the values of the geometrical parameters of the real robot differ by a limited amount from their theoretical values. Hence we are looking not for values for the geometrical parameters but for ranges $\mathcal{I}_{P_j}$ for each of them such that they satisfy the property:

$$\begin{aligned} \forall\ P_j\ \in\ \mathcal{I}_{P_j},\quad j\ \in [1,26]\ \textbf{and}\ \forall\ \mathbf{X}\ \in\ \mathcal{W} \\ |J_i(\mathbf{X}, \mathcal{P})|\ \in\ \Delta\mathbf{X}_d^i \end{aligned} \tag{3}$$

II. Theoretical analysis

A. Dealing with manufacturing tolerances

As mentioned previously we intend to find the solution of the synthesis problem such that even with manufacturing errors the property (3) will be satisfied. We may assume that the manufacturing tolerances on the geometrical parameters are bounded and that their bounds are known. If P_j^m is used as nominal value of a given geometrical parameters P_j for the manufacturing process we may assume that the real value of P_j will lie in the range $[P_j^m - \epsilon_j, P_j^m + \epsilon_j]$.

This implies that if we find a solution interval $\mathcal{I}_{P_j} = [a, b]$ for the parameter P_j whose width is larger or equal to $2\epsilon_j$, then we are able to guarantee that the real robot will satisfy property (3) by choosing as theoretical manufacturing value a number in the range $[a - \epsilon_j, b - \epsilon_j]$ as this guarantee that the real value will be in $\mathcal{I}_{P_j}$.

B. Dealing with the jacobian matrix

A main difficulty of this problem is that in general for parallel robots the matrix J has an unknown analytical form (or at least a very complex one, that is useless) while its inverse J^{-1} is perfectly known. Indeed for the Gough platform the i-th row of J^{-1} is:

$$J_i^{-1} = ((\frac{\mathbf{A_iB_i}}{||\mathbf{A_iB_i}||}\ ,\ \frac{\mathbf{CB_i} \times \mathbf{A_iB_i}}{||\mathbf{A_iB_i}||})) \tag{4}$$

where $||\mathbf{A_iB_i}||$ is the Euclidean norm of the vector $\mathbf{A_iB_i}$ or, in other words, the length ρ_i of leg i.

Consider now the linear system

$$J^{-1}(\mathcal{P}, \mathbf{X})\Delta\mathbf{X} = \Delta\rho \tag{5}$$

The problem we intend to solve is to find interval values for each parameters such that by choosing any value for the parameters in these intervals the linear system will be such that for all $\mathbf{X}$ in $\mathcal{W}$ and all values of $\Delta\rho_i$ in [-1,1] all the solutions $\Delta\mathbf{X}$ of the system lie in the range $\Delta\mathbf{X}_d$.

If all the geometry and pose parameters lie within given ranges, then interval analysis allows to determine a range for each element of J^1 that include all possible values for the elements (even taking into numerical round-off errors). The drawback of interval analysis is that these ranges will usually be overestimated with a negative impact on the efficiency as we will examine if *all* the systems defined by the interval matrix have a solution included in $\Delta\mathbf{X}_d$. However we may limit this overestimation by using classical methods of interval analysis, such as the use of the derivatives of the elements with respect to the geometry and pose parameters [5], [6].

The interval evaluation $J_{\mathcal{I}}^{-1}$ of J^{-1} will be computed using the procedure `Compute_Jacobian`$(P_{\mathcal{I}}, \mathbf{X}_{\mathcal{I}})$ that takes as input the ranges $P_{\mathcal{I}}, \mathbf{X}_{\mathcal{I}}$ for the geometry and pose parameters.

Note an interesting property of the system (5). Each element J_{ij}^{-1} of the i-th line of J^{-1} may be written in the form U_{ij}/ρ_i. Multiplying both sides of the system (5) by the diagonal matrix A_ρ whose diagonal elements are ρ_i leads to the system:

$$J_m^{-1}(\mathcal{P}, \mathbf{X})\Delta\mathbf{X} = A_\rho\Delta\rho \tag{6}$$

where J_m^{-1} is the matrix $((U_{ij}))$. This system has hence the same solutions than (5). This system is interesting for us as the derivatives of U_{ij}, ρ_i with respect to the unknowns are relatively simple while the derivative of U_{ij}/ρ_i may be quite complex. Hence the interval evaluation of the matrix J_m^{-1} will suffer from a lower overestimation compared to the interval evaluation of J^{-1}.

C. Solving interval linear systems

Finding the solutions $\mathbf{Y}$ of a linear system:

$$\mathbf{AY} = \mathbf{b} \tag{7}$$

where $\mathbf{A}$ is an interval matrix and $\mathbf{B}$ an interval vector is a classical problem in interval analysis.

We want to determine the solution set defined by:

$$\Sigma_{\exists,\exists}(\mathbf{A}, \mathbf{b}) = \{y|, \exists A \in \mathbf{A}, \exists b \in \mathbf{b}, A.y = b\}, \tag{8}$$

To determine $\Sigma_{\exists,\exists}(\mathbf{A}, \mathbf{b})$ or only the tightest enclosing box is an NP-hard problem and hence expensive in high

dimension as its shape can be quite complicated. However, it is possible to find a box enclosure of $\Sigma_{\exists,\exists}(\mathbf{A},\mathbf{b})$ by an interval vector $\mathbf{Y_S}$ with limited overestimation, provided that the intervals are narrow enough.

The numerical conditioning of the system (7) is essential for an efficient solving. This quality may be improved if the matrix $\mathbf{A}$ is *diagonally dominant* (see [11]) i.e. roughly if this matrix is close to the identity matrix. That is possible by *preconditioning* the system i.e. by multiplying both terms of equation (7) by a conditioning scalar matrix $\mathbf{M}$. Indeed the system becomes $\mathbf{MAY}^{'} = \mathbf{Mb}$ ($\Sigma_{\exists,\exists}(\mathbf{A},\mathbf{b}) \subset \Sigma_{\exists,\exists}(\mathbf{M}.\mathbf{A},\mathbf{M}.\mathbf{b})$ but often $\mathbf{Y} \subset \mathbf{Y}^{'}$ – see [5]). Usually the best conditioning matrix is the matrix obtained by taking the middle point of all the interval elements of $\mathbf{A}$ then by inverting it : $\mathbf{M} = [Mid(\mathbf{A})]^{-1}$.

The basic method to provide the solution of (7) is an interval adaptation of the *Gauss-elimination method* which computes $\mathbf{Y}$ without the need of an initial estimation of $\mathbf{Y}$. However if such initial estimation is available, it is possible to apply an iterative fixed point algorithm such as the *Interval Gauss-Seidel scheme* given by

$$\mathbf{Y}_{k+1} \leftarrow \mathbf{Y}_k \cap \mathbf{C}^{-1}(\mathbf{b} - \mathbf{D}.\mathbf{Y}_k)$$

with $\mathbf{C} = Diag(\mathbf{A})$ and $\mathbf{D} = \mathbf{A} - \mathbf{C}$.

Another iterative method, the *Krawczyk iterative scheme* may be used. It is defined as

$$\mathbf{Y}_{k+1} \leftarrow \mathbf{Y}_k \cap \mathbf{b} + (I - \mathbf{A}).\mathbf{Y}_k$$

but numerous other methods may also be applied [6].

For the application at hand we will use an interesting propriety of those iterative algorithms: if $\mathbf{Y}_{k+1} \subset \mathbf{Y}_k$ then all the solutions of all the linear systems are included in $\mathbf{Y}_{k+1}$, [11]. By applying these methods on the system (5), we are able to calculate an interval evaluation of the solutions $\Delta\mathbf{X}$ and if this evaluation is included in $\Delta\mathbf{X}_d$, then property (3) is satisfied for any geometry and pose parameters within their respective range. We are thus able to design a procedure `Linear_Solve`$(J_{\mathcal{I}}^{-1}, P_{\mathcal{I}}, \mathbf{X}_{\mathcal{I}})$ that will return 1 if all the solutions of (5) are included in $\Delta\mathbf{X}_d$ whatever are the values of $P_i, \mathbf{X}_i$ in their range $P_{\mathcal{I}}, \mathbf{X}_{\mathcal{I}}$. Otherwise, the procedure will return 0.

Note that all the methods that are used to find the solutions set of a linear system are very sensitive to the width of the interval of the elements of the $\mathbf{A}$ matrix. Hence it is interesting to use in `Linear_Solve` the matrix J_m^{-1} of the system (6) instead of the matrix J^{-1} of (5). Another reason to use J_m^{-1} instead of J^{-1} is that most of the methods used to determine the solution of a linear system involve at some point the calculation of the ratio A_{ij}/A_{ii}. In our case A_{ij}, A_{ii} are written as $U_{ij}/\rho_i, U_{ii}/\rho_i$ and clearly the interval evaluation of U_{ij}/U_{ii} will be always less overestimated than $(U_{ij}/\rho_i)/(U_{ii}/\rho_i)$. See section III-B for further details.

III. The algorithm

A. Algorithm principle

We present here an outline of the algorithm. As usual for an interval analysis based method we define a *box* $\mathcal{B}$ as a set of ranges, one for each of the geometry and pose parameters. The algorithm will create and discard boxes that are stored in a list $\mathcal{L}$. Initially this list has only one element $\{\mathcal{R}_P, \mathcal{W}\}$ and the i-th box in the list will be denoted $\mathcal{B}_i$ while the total number of boxes in $\mathcal{L}$ will be n. A box $\mathcal{B}_j$ may be *bisected* using the `Bisection`$(\mathcal{B}_j)$ procedure. In this procedure we consider the variable P_k that has the largest interval. Its interval $\mathcal{I}^k = [a_k, b_k]$ is bisected at its middle point in order to create 2 new intervals $\mathcal{I}_1^k = [a_k, (a_k + b_k)/2]$, $\mathcal{I}_2^k = [(a_k + b_k)/2, b_k]$. The bisection process on $\mathcal{B}_j$ results in 2 new boxes that have the same ranges than $\mathcal{B}_j$ except for the variable k, one of the box having for this variable the range $\mathcal{I}_1^k$ while the other has $\mathcal{I}_2^k$.

The algorithm will process the boxes in $\mathcal{L}$ in sequence, l being the number of the box that is currently processed and the algorithm starts with $l = 1$.

The algorithm proceeds along the following steps:

1) if $l > n$ then `EXIT`
2) J_l=`Compute_Jacobian`$(\mathcal{B}_l)$
3) if `Linear_Solve`$(J_l, \mathcal{B}_l)$=1, then $\mathcal{B}_l$ is solution
   a) store $\mathcal{B}_l$ in the result file, $l = l + 1$, go to 1
4) if the widths of all the ranges in $\mathcal{B}_l$ are lower than $2\epsilon_j$ then $l = l + 1$, go to 1
5) `Bisection`$(\mathcal{B}_l)$: the 2 new boxes created by this procedure are stored in $\mathcal{L}$ at position $n+1$ and $n+2$. $n = n + 2$, $l = l + 1$, go to 1

This algorithm is straightforward:

- solution are determined at step 3
- boxes that may contain solution but which are too small to ensure that a physical instantiation will be enclosed in the solution are eliminated at step 4
- the algorithm will stop when reaching step 1 when all boxes in the list have been processed

Note that we may take into account constraints on the geometry parameters during the computation by introducing a *filtering* procedure before step 2. For example we may want that the attachment points on the platform and on the base are at a minimal distance from each other to avoid interference problems. Accordingly we may design a procedure that calculate an interval evaluation of the distances between each pair of attachment points. If the upper bound of the evaluation is lower than the distance threshold the box will be discarded. If the upper bound is greater than the threshold, then the box will be bisected.

The described algorithm returns as result an approximation of the set of all possible solutions of the synthesis problem. It must be noted that, as all interval analysis based method, it may be implemented using a distributed approach i.e. using a set of computers: a master program will manage the list $\mathcal{L}$ and send a box to process to a free slave computer $\mathcal{S}$. This slave computer program execute a few steps of the algorithm with its own boxes list $\mathcal{L}_\mathcal{S}$ until either $\mathcal{L}_\mathcal{S}$ is exhausted or that the number of boxes in $\mathcal{L}_\mathcal{S}$ has reached a given threshold. Then the slave computer will return to the master program the list $\mathcal{L}_\mathcal{S}$ (possible empty)

that has to be processed together with another set (also possibly empty) of synthesis solutions. The master program will include $\mathcal{L}_{\mathcal{S}}$ in $\mathcal{L}$ and will send another box to process to $\mathcal{S}$.

Failure of the algorithm may occur if the components of the inverse jacobian matrix have a very complex form. Indeed interval analysis will usually overestimate the ranges for these components and the size of this overestimation increase with the complexity of the analytical form of the components. A consequence of this overestimation is that the `Linear_Solve` procedure may fail to determine if all solutions of the linear systems are included in $\Delta\mathbf{X}_d$ even if the size of the ranges for the geometry and workspace parameters is small.

Another possible cause of failure is that we do not take into account the dependency of the components of the inverse jacobian matrix. Indeed the interval matrix $\mathbf{A}$ that will be considered by `Linear_Solve`.describes a larger set of linear systems than the one that we will get by calculating the matrix for all values of the parameters in their range. But taking into account the dependency between the elements of $\mathbf{A}, b$ is difficult.

In the following sections we present possible improvements of the method.

B. Improvement of the Gauss elimination scheme

Let us assume that we have an $n \times n$ interval linear system:

$$\mathbf{A}(\mathbf{X}).Y = b(\mathbf{X})$$

where the $\mathbf{A}, b$ elements are function of the unknowns $\mathbf{X}$. When the unknowns lie in given ranges we may compute an interval evaluation $\mathbf{A}^{(0)}$ of $\mathbf{A}$ and an interval evaluation $b^{(0)}$ of b (possibly using the derivatives of the components of $\mathbf{A}, b$ to improve these interval evaluations). The Gauss elimination scheme may be written as [10]

$$A_{ik}^{(j)} = A_{ik}^{(j-1)} - A_{ij}^{(j-1)} A_{jk}^{(j-1)} / A_{jj}^{(j-1)} \quad \forall\ i,\ j > k \quad (9)$$

$$b_i^{(j)} = b_i^{(j-1)} - A_{ij}^{(j-1)} b_j^{(j-1)} / A_{jj}^{(j-1)} \quad (10)$$

The enclosure of the variable Y_j can then be obtained from $Y_{j+1}, \ldots, Y_n$ by

$$Y_j = (b_j^{(j-1)} - \sum_{k>j} A_{jk}^{(j-1)} Y_k) / A_{jj}^{(j-1)} \quad (11)$$

The important point is that in the above equations appear the product and ratio of elements that are not independent in our problem. Hence the enclosure solution we get does not take into account this dependency.

Note that by using classical derivation rules an interval evaluation of the derivatives of the elements at iteration j i.e. $A_{ik}^{(j)}, b_i^{(j)}, Y_j$ may be calculated as soon as the derivatives of the elements having the superscript $(j-1)$ are available. As the derivatives at iteration (0) are simply the derivatives of the components of $\mathbf{A}, b$ that are available we may therefore compute the derivatives of all the elements at iteration (j) and use these derivatives to improve their interval evaluations. Our experiments has shown that the use of derivatives may reduce the size of the solution enclosure by a 5 to 90 %.

C. Improvement based on workspace bisection

The above algorithm may fail if the considered workspace is large. Indeed in that case the ranges for the components of the inverse jacobian matrix may be quite large even for small ranges for the geometry parameters, hence leading to a failure of the `Linear_Solve` procedure. To avoid this problem we may consider bisecting also the workspace parameters as soon as the ranges for the geometry parameters are small enough. We will have a list of boxes (to avoid any ambiguity we will use the notation *workspace box* in that case) that will be submitted to the `Linear_Solve` procedure. All workspace boxes that satisfy property (3) are eliminated from the list, the other one being bisected. Only a limited number of bisection is allowed: if this number is reached we come back to the main algorithm dealing with the geometry parameters. Hence we design a workspace procedure returns 1 if for all the boxes in the list property (3) has been verified and 0 otherwise. This procedure may be used either before or after step 4 of the above algorithm. It is relatively computer intensive and hence should be carefully used: in our implementation we use it only when a given number of geometry parameter ranges have a width such that they will be no more bisected.

Note that the workspace algorithm allows one to deal with more complex workspace than the hyper-cube we have been considering up to now as soon as we are able to design a test that allows one to determine if a box is either fully inside the workspace or fully outside the workspace. In the later case the workspace box will be rejected from the list while in the former case the accuracy within the box will be considered.

Assume for example that the workspace for C is a sphere centered at $S_1(x_1, y_1, z_1)$ with radius l_1 and that the ranges for x_c, y_c, z_c in the workspace box are X_c, Y_c, Z_c. We compute the interval evaluation of $X_c - x_1, Y_c - y_1, Z_c - z_1$ and denote by $\underline{X}, \overline{X}$ the lower and upper bound of the interval X. Then the interval evaluation of the square of the distance between a point in the workspace box and S_1 is $(X_c - x_1)^2 + (Y_c - y_1)^2 + (Z_c - z_1)^2$ and:

- if $\overline{(X_c - x_1)^2} + \overline{(Y_c - y_1)^2} + \overline{(Z_c - z_1)^2}$ is lower than l_1^2, then the workspace box is fully inside the sphere
- if $\underline{(X_c - x_1)^2} + \underline{(Y_c - y_1)^2} + \underline{(Z_c - z_1)^2}$ is greater than l_1^2, then the workspace box is fully outside the sphere

If none of these two conditions are satisfied, then a part of the workspace box is inside the sphere while its complementary is outside the sphere (indeed in that case as there is only one occurrence of the unknown x_c, y_c, z_c in the squared distance expression, the interval evaluation of the squared distance is exact: there are points in the workspace box whose squared distance to S_1 is exactly either the lower or upper bound of the interval evaluation). In that case we will just use the `Linear_Solve` procedure to check if the workspace box satisfy the property (3). If this is the case we discard the workspace box otherwise it will be bisected.

D. Improvement based on Oettli theorem

We present here an approach that may allow to determine larger solution boxes than with `Linear_Solve`. Assume that at the middle point of a box (i.e. for a given robot geometry) and at the center of the workspace the robot accuracies are included in $\Delta\mathbf{X}_d$ while for some other geometries within the box and some points of the workspace at least one of the accuracies that is not included in $\Delta\mathbf{X}_d^i$. As the solutions of (5) are continuous functions of the geometry and workspace parameters there must exist points within the box where at least one of the extremal accuracy is exactly the upper bound of $\Delta\mathbf{X}_d^i$ while the other accuracies lie in $\Delta\mathbf{X}_d^j$. If it is possible to prove that no such point exists, then the box is a solution box. For this proof we will rely on Oettli theorem [14]: let

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{12}$$

be a set of linear systems with $\mathbf{A}$ an interval matrix and $\mathbf{b}$ an interval vector. Let $\mathbf{A_c}$ be the matrix obtained by taking the mid-point of the range of the components of $\mathbf{A}$ and $\mathbf{b_c}$ be the mid-vector of $\mathbf{b}$. We define Δ as the matrix whose components are the half diameter of the ranges of the components of $\mathbf{A}$ and similarly δ as the vector whose components are half the diameter of the range of $\mathbf{b}$. Oettli theorem states that there are systems $Ax = b$ with A in $\mathbf{A}$, x in $\mathbf{X}$ and b in $\mathbf{b}$ that have solutions if and only if:

$$|\mathbf{A_c}x - \mathbf{b_c}| \leq \Delta|x| + \delta \tag{13}$$

We will use this theorem on (5) to determine if there is a system that admit a solution with one of its component x_i being fixed to $\Delta\mathbf{X}_d^i$ while the others may have any value in $\Delta\mathbf{X}_d^j$. As for $\mathbf{b}$ it may be any of the 64 vectors that have +1 or -1 as entries. If we show that (13) is never satisfied whatever is x and for all i in [1,6], then we have proven that we have a solution box. The computation amounts to verify 384 inequalities of type (13) (64 inequalities corresponding to the different combination for $\mathbf{b}$, this for each of the 6 components of x that are fixed to their extremal value). Verifying (13) is a complex issue but may be done using a bisection approach as the 5 unknowns in x (the components of x that are not the i-th component) have bounded values. As we will see in the example this computation is intensive but allows one to obtain larger solution boxes.

IV. Implementation and results

The previous algorithms have been implemented using the `BIAS/Profil` interval arithmetics package (that implement basic operations of interval arithmetics) and the C++ library `ALIAS` that implement high-level interval analysis procedures such as the bisection, the linear system solver or the interval evaluation using the derivatives. A Maple interface to `ALIAS` allows one to produce automatically the C++ code that is necessary for the interval evaluation of the U_{ij}, ρ_i quantities, together with their derivatives.

As the result of the algorithm is a list of possible ranges for the 26 geometry parameters it is impossible to represent graphically the full result. Hence we will just present a simple example in which the θ, α angles are supposed to have a manufacturing tolerances of 0.001 rad while their ranges have exactly 2 times this tolerance (thereby these parameters will not be touched by the bisection process) and the attachment points A, B are supposed to be perfectly coplanar.

The parameters R_1, r_1 have a tolerance of 0.001 and their initial range is [193,195] and [9,10]. The workspace is small with a range of [-1,1] for x_c, y_c and [199,201] for z_c while the pitch, yaw, roll angles have a range of [-0.005,0.005] rad. The desired accuracy is $\pm$ 10 for $\Delta x_c, y_c, z_c$ and $\pm$ 1 for the angular errors. The smaller the radius of the platform is, the closer we are to a singular configuration. Hence the amplification factor between the leg lengths measurements and the platform positioning error will be high if the platform radius is small.

If we do not use the improved Gauss elimination scheme described in section III-B the result is a list of 819 solution boxes for a total volume of 2.0235e-33. The computation time for establishing this result is about 34mn on a DELL D400, 1.7 Ghz. With the improved Gauss elimination scheme we get 677 solution boxes for a total volume of 7.663e-33 in a computation time of 1 hour and 7 mn. Both results are presented on figure 2.

Fig. 2. In grey the allowed region for R_1, r_1: for all these base and platform radius the desired accuracy is reached (the darker area is the result obtained without the improved Gauss elimination scheme).

The computation time is relatively large but it must be understood that the result presents guaranteed solutions to the synthesis problem. It may be decreased by using a distributed implementation although we believe that improvements of the algorithm are still possible. Solutions are obtained almost immediately and most of the computation is spent in the workspace procedure when trying to determine if a box is solution of the synthesis problem. Hence we cannot claim to obtain all the synthesis solution unless we allow for a computer intensive use of the workspace procedure.

A large number of boxes may be produced (simply because there may be a large number of possible design solutions), but file storage is not a problem. Furthermore we may reduce this number by using the output of the algorithm as an input for another algorithm that will deal with another performance index (for example the reachable workspace [9]) and that will eliminate some of the boxes.

V. Error analysis

Assume now that we have determined a geometry for the robot, that satisfies property (3). We want now to proceed to an error analysis of this robot i.e. we want to determine what are the extremal positioning errors $\Delta\mathbf{X}_r$ over the prespecified workspace (indeed at this stage we only know that $\Delta\mathbf{X}_r$ is included in $\Delta\mathbf{X}_d$). But the objective is also to take into account the manufacturing errors i.e. we want to determine the extremal positioning errors whatever are the geometric parameters of the real robot.

A simple variant of the previous algorithm allows one to determine this extremal positioning errors up to an arbitrary accuracy μ. The main idea is to compute the positioning error for the middle point of the box. Using this calculation we are able to update a set of current values $\Delta\mathbf{X}^M$ for the extremal positioning errors. Then `Linear_Solve` is used with $\Delta\mathbf{X}^M \pm \mu$ as enclosure for the solutions. If this procedure returns 1, then there is no value of $\Delta\mathbf{X}$ greater in absolute value than $|\Delta\mathbf{X}^M + \mu|$ in the current box: this box can thus be discarded. Note that this is the only case for which a box is discarded as step 4 of the algorithm is no more used. After running this algorithm $\Delta\mathbf{X}_r$ is guaranteed to be included in $[\Delta\mathbf{X}^M - \mu, \Delta\mathbf{X}^M + \mu]$.

As an example the first box of the previous result has been examined with $R_1 = 194.9375$, $r_1 = 9.8125$ using $\mu = [0.5, 0.5, 0.5, 0.05, 0.05, 0.05]$. We have found out that the maximal positioning errors were $\pm$ [7.67256 ,3.5064 ,2.3123 ,0.306632, 0.815007 ,0.460367] in about 45 seconds. If we change μ to $\mu = [0.2, 0.2, 0.2, 0.02, 0.02, 0.02]$ we get the same result in 1846s.

VI. Conclusion

Synthesis of parallel manipulator with respect to accuracy requirements is a difficult problem. The interval analysis-based approach we have proposed allows one to solve this problem with the additional advantages of providing not only one solution but a continuous set that allows one to take into account manufacturing errors. Note that this approach may be used for other mechanisms than parallel robot as soon as there exists a mean to calculate an interval evaluation of the elements of the Jacobian matrix.

Like most algorithms using interval analysis although it principle is fairly simple its implementation requires a lot of expertise to be efficient. For that purpose we have reported the use of the derivatives in the Gauss elimination scheme (this use is presented here for the first time to the best of our knowledge) that allows one to obtain larger solution boxes. The main reason for the large computation time is that we have a procedure to test if a box *is a solution* but the only criteria to determine if a box *is not a solution* is based on its overall size. The lack of such negative test prohibits us to use various methods of interval analysis that allows to filter boxes.

Our prospective are to develop:

- algorithms that will allow to determine that a whole box is not a solution
- algorithms that take more into account the dependency between the components of the J^{-1} matrix
- algorithms to improve the speed of the error analysis. A possibility is to use a fast local optimization procedure as soon as it has been determined that some elements of the accuracy at the middle point of a box are larger than the current maximal value

References

[1] Brisan C., Franitza D., and Hiller M. Modelling and analysis of errors for parallel robots. In *1st Int. Colloquium, Collaborative Research Centre 562*, pages 83–96, Braunschweig, May, 29-30, 2002.

[2] Di Gregorio R. and Parenti-Castelli V. Geometric error effects on the performances of a parallel wrist. In *3rd Chemnitzer Parallelkinematik Seminar*, pages 1011–1024, Chemnitz, April, 23-25, 2002.

[3] Han C. and others . Kinematic sensitivity analysis of the 3-UPU parallel manipulator. *Mechanism and Machine Theory*, 37:787–798, 2002.

[4] Han C-S., Hudgens J.C., Tesar D., and Traver A.E. Modeling, synthesis, analysis and design of high resolution micromanipulator to enhance robot accuracy. In *IEEE Int. Workshop on Intelligent Robot and Systems (IROS)*, pages 1153–1162, Osaka, November, 3-5, 1991.

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

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

[7] Kim H.S. and Choi Y.J. The kinematic error bound analysis of the Stewart platform. *J. of Robotic Systems*, 17(1):63–73, 2000.

[8] Masory O., Wang J., and Zhuang H. On the accuracy of a Stewart platform-part II: Kinematic calibration and compensation. In *IEEE Int. Conf. on Robotics and Automation*, pages 725–731, Atlanta, May, 2-6, 1993.

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

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

[11] Neumaier A. *Introduction to Numerical Analysis*. Cambridge Univ. Press, 2001.

[12] Parenti-Castelli V. and Di Gregorio R. Influence of manufacturing errors on the kinematic performance of the 3-UPU parallel mechanism. In *2nd Chemnitzer Parallelkinematik Seminar*, pages 85–99, Chemnitz, April, 12-13, 2000.

[13] Patel A.J. and Ehmann K.F. Volumetric error analysis of a Stewart platform based machine tool. *Annals of the CIRP*, 46/1/1997:287–290, 1997.

[14] Rohn J. Systems of interval linear equations and inequalities (rectangular case). Research Report 875, Institute of Computer Science, Academy of Sciences of the Czech Republic, September 2002.

[15] Ropponen T. and Arai T. Accuracy analysis of a modified Stewart platform manipulator. In *IEEE Int. Conf. on Robotics and Automation*, pages 521–525, Nagoya, May, 25-27, 1995.

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

[17] Tischler C.R. and Samuel A.E. Predicting the slop of in-series/parallel manipulators caused by joint clearances. In *ARK*, pages 227–236, Strobl, June 29- July 4, 1998.

[18] Wang J. and Masory O. On the accuracy of a Stewart platform-part I: The effect of manufacturing tolerances. In *IEEE Int. Conf. on Robotics and Automation*, pages 114–120, Atlanta, May, 2-6, 1993.