

A Generic Trajectory Verifier for the Motion Planning of Parallel Robots

Jean-Pierre Merlet

INRIA Sophia-Antipolis
BP 93 06902 Sophia-Antipolis, France
e-mail: Jean-Pierre.Merlet@sophia.inria.fr

In this paper we consider the problem of trajectory verification for a classical Gough-Stewart platform i.e. we want to verify if a given trajectory obeys various criteria which define its validity, for example that the trajectory lies fully inside the workspace of the robot and is singularity-free. We propose an almost real-time method that may deal with almost any trajectory and any validity criterion and can manage uncertainties on the specified trajectory, for example to take into account control errors.

[DOI: 10.1115/1.1401735]

1 Introduction

Off-line verification of a trajectory of a parallel robot is very important in practical applications, for example when using such a machine for manufacturing operations. Indeed it is necessary to check if the trajectory is *valid* i.e. that it verifies a set of criteria, that will be called the *validity criteria*. Examples of validity criteria are:

- the whole trajectory must lie inside the reachable workspace of the robot
- the minimum of the dexterity of the robot on the trajectory should not be lower than a fixed threshold.
- the absolute value of the articular forces should not exceed a fixed threshold

There are numerous possible validity criteria while various types of trajectories may have to be checked, from straight lines and arcs of circles to quite complex trajectories such as shown in Fig. 1. We will assume that the trajectories are 6-dimensional i.e. both the location and orientation of the end-effector are time-dependent.

Surprisingly few papers in the literature are devoted to the problem of trajectory verification and motion planning for parallel robots. Singularity-free path planning has been addressed in [1,2] while path planning was addressed from a control view point in [3]. An algorithm for checking the validity of a trajectory composed of straight line segments has been proposed in [4] and this paper is an extension of this algorithm.

An alternative approach is to *design* the robot in such way that it verifies some validity criteria by design. For example it is possible to determine the robot geometry so that its workspace includes a specified workspace [5,6,7,8,9] and then to check that this workspace is singularity-free [10,11,12,13]. But this approach is quite complex and for the time being it can be used for few validity criteria.

The purpose of this paper is to present an algorithm that enables one to check almost any type of trajectory and set of validity criteria, even very complex ones, taking into account the fact that the trajectory followed by the robot may be slightly different from the specified one due to control errors.

For the sake of simplicity we will present our algorithm for the classical Gough-type parallel manipulator [14] illustrated in Fig. 2, although our algorithm may be used for almost any type of parallel machine. In this robot a base and a platform are connected through 6 extensible legs which have a ball-and-socket joint at

each extremity A_i, B_i . Linear actuators enable to change the leg lengths which in turn enable to control the position and orientation of the platform.

We define a reference frame $O, (x, y, z)$ which is attached to the base and a mobile frame $C, (x_c, y_c, z_c)$ which is attached to the platform. We may represent a pose of the platform by the coordinates x_c, y_c, z_c of the origin C of the mobile frame in the reference frame and its orientation by using the classical Euler angles ψ, θ, ϕ which enable to define a rotation matrix R for transforming the coordinates of a vector expressed in the mobile frame into its coordinates in the reference frame. The coordinates of the attachment points A_i are supposed to be known in the reference frame, while the coordinates of the B_i points are known in the mobile frame.

2 Trajectory and Validity Criteria

We will assume that a trajectory of the platform is specified by defining the parameters of the pose of the platform as analytical functions of the time T , assumed to lie in the range $[0,1]$, i.e. that we have

$$x_c = D_x(T) \quad y_c = D_y(T) \quad z_c = D_z(T) \quad (1)$$

$$\psi = D_\psi(T) \quad \theta = D_\theta(T) \quad \phi = D_\phi(T) \quad (2)$$

2.1 Workspace Constraints. A first validity criterion can be defined immediately: in practice the leg lengths must lie within some ranges that will be taken as identical for all legs and will be denoted $[\rho_{\min}, \rho_{\max}]$ and therefore a valid trajectory must verify at least this constraint.

For a Gough platform the length ρ of a leg is simply the norm of the vector \mathbf{AB} which may be written as

$$\rho^2 = \|\mathbf{AB}\|^2 \quad \mathbf{AB} = \mathbf{AO} + \mathbf{OC} + \mathbf{CB} \quad (3)$$

For a given robot the first element of the right-hand term of \mathbf{AB} is known. The last element, \mathbf{CB} is equal to $R\mathbf{CB}_r$ where R is the rotation matrix, a function of ψ, θ, ϕ , and \mathbf{CB}_r is the known coordinates vector of B in the mobile frame.

Using Eqs. (1), (2) we may transform Eq. (3) into a time function. Verifying that ρ lies in the range $[\rho_{\min}, \rho_{\max}]$ for *any* T in the range $[0,1]$ is not a simple problem. For example, due to the non-linearity, the end-points of the trajectory (i.e. the pose at $T=0$ and $T=1$) may lie inside the workspace although one or more points of the trajectory may be outside this workspace.

Mechanical limits on the passive joint located at A_i, B_i may also lead to an infeasible trajectory. For example a link connected to a ball-and-socket joint is usually constrained to lie within a cone whose apex is the center of the joint: hence the angle be-

Contributed by the Mechanisms Committee for publication in the JOURNAL OF MECHANICAL DESIGN. Manuscript received January 2000. Associate Editor: G. S. Chirikjian.

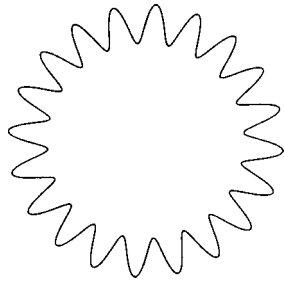


Fig. 1 An example of complex trajectory

tween the link and the cone axis (the *main direction* of the joint) defined by the unit vector n_i must be lower than a given value α . Therefore we must have

$$\frac{\mathbf{A}_i \mathbf{B}_i \cdot n_i}{\|\mathbf{A}_i \mathbf{B}_i\|} > \cos \alpha$$

Using Eqs. (1), (2) we may transform this inequality into a time-dependent inequality $\mathcal{G}(T)$ such that for a valid trajectory the condition $\mathcal{G}(T) > 0$ must be verified for all T in $[0,1]$. Another interesting validity criterion is to check that there is no interference between the legs. If the legs are cylinders it was shown in [4] that leg interference is avoided if a set of time-dependent inequalities are verified.

3 Dexterity Constraint

It is well known that the accuracy $\Delta \mathbf{X}$ on the positioning of the platform is linearly related to the accuracy on the length measurements $\Delta \rho$ by:

$$\Delta \rho = J^{-1} \Delta \mathbf{X} \quad (4)$$

$$\Delta \mathbf{X} = J \Delta \rho \quad (5)$$

where J is the jacobian matrix of the robot. If we assume that the accuracy is identical for all the leg sensors, the leg length measurement errors lie inside a hyper-cube. The corresponding errors for the generalized coordinates is a hyper-polyhedra that is obtained by mapping the length errors hyper-cube through Eq. (5). To determine the amplification factor between both errors it is necessary to quantify the shape of the hyper-polyhedra through a *dexterity index* which may be, for example, the *manipulability index* $\sqrt{|J^{-1}|}$ [15] or the inverse of the condition number κ of J^{-1} which is defined as the ratio between the smallest singular value and the largest one. Both dexterity indices will be 0 at a *singular-*

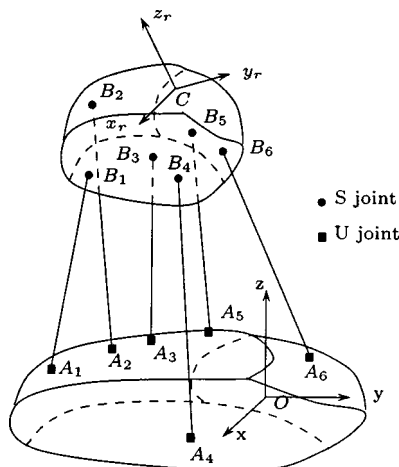


Fig. 2 The classical Gough-type parallel robot

ity of the robot which is defined by $|J^{-1}| = 0$. Singularities have to be avoided as motion around a singularity will be inaccurate and as very large forces in the legs occur for a pose of the platform in their vicinity.

To avoid singularities on the trajectory we will thus impose a minimal threshold ϵ on the dexterity index \mathcal{D} , i.e. $\mathcal{D} > \epsilon$. We will assume that we are able to calculate an analytical form of \mathcal{D} and using Eq. (2) we may transform \mathcal{D} into a time-dependent function $\mathcal{D}(T)$: a valid trajectory will observe the condition $\mathcal{D}(T) > \epsilon$ for all T in $[0,1]$.

4 The Motion Verifier

Our purpose is to design an algorithm which enables verification if a given trajectory is fully inside the workspace and if the dexterity criterion at any point on the trajectory is not lower than a fixed threshold ϵ . Additionally we will assume that another set of n validity criteria $\mathcal{G}_i(T)$ has been defined. Therefore a trajectory will be valid if:

$$\rho_{\min} \leq \rho_i(T) \leq \rho_{\max} \quad \text{for } i = 1, \dots, 6 \quad (6)$$

$$\mathcal{G}_i(T) > 0 \quad \text{for } i = 1, \dots, n \quad (7)$$

$$\mathcal{D}(T) > \epsilon \quad (8)$$

for all T in $[0,1]$.

4.1 Calculation of the Analytical Form of the Constraints.

Our algorithm will make extensive use of the analytical form of the constraints that will be obtained through symbolic computation. We will first consider that the description of the geometry of the robot, i.e. the location of the A_i , B_i points, and the minimum and maximum leg lengths, are available in a file, called the *robot file*. Then, we will use MAPLE to get the analytical form of the constraints and validity criteria (in the following sections we will assume that the reader is familiar with MAPLE). The user will define its trajectory in a *trajectory file* using MAPLE syntax and some notation conventions. For example the pose parameters x_C , y_C , z_C will be represented by the MAPLE symbols x , y , z , while the orientation angles ψ , θ , ϕ are represented by the symbols p , t , h . Hence, for example, the following MAPLE file:

```
x:=3*sin(2*Pi*T): y:=3*cos(2*Pi*T): z:=56: p:=0:
```

```
t:=5*Pi/180: h:=0:
```

will describe that the trajectory is an horizontal circle centered at point (0,0,56) with radius 3 while the orientation angles are equal to $\psi=0$, $\theta=5$ and $\phi=0$ degree.

Another example is the gear trajectory shown in Fig. 1 for which the trajectory file is:

```
p:=0:t:=0:h:=0:z:=56:x:=(3+0.5*sin(40*Pi*T))*sin(2*Pi*T):
```

```
y:=(3+0.5*sin(40*Pi*T))*cos(2*Pi*T):
```

As soon as the robot file has been specified our program will read the coordinates of the A_i , B_i points and create a temporary MAPLE file data that describes the coordinates. The user may also include additional validity criteria by writing in a file, called the *validity criteria file*, some MAPLE procedures that allow to calculate the analytical form of the validity criteria \mathcal{G}_i . As soon as the data file, the trajectory file trajectory and the validity criteria file have been defined the program will run the following specific MAPLE program:

```
read("data");
read("trajectory");
#compute rhoi as a time function
#compute Gi as a time function
#compute D as a time function
```

The purpose of this program is to obtain the analytical form of the validity criteria. These analytical expressions will be written in specific files for later use. For example if the trajectory is the

horizontal circular trajectory presented above and the robot has the geometry defined in the Appendix the analytical form of the squared length of the first leg is:

$$3311 + 784 * \sin(\text{Pi}/36) - 126 * \cos(\text{Pi}/36) + (42 * \cos(\text{Pi}/36) - 54) * \cos(2 * \text{Pi} * T) + 36 * \sin(2 * \text{Pi} * T)$$

4.2 Interval Analysis. Our algorithm will require the computation of lower and upper bounds of all the quantities defined in the previous section for a given range for T . For example, being given a range on T we must be able to compute two reals a, b with $a < b$ such that $a \leq \mathcal{D}(T) \leq b$ for any value of T in its range. Note that it will not be necessary to get sharp bounds on the quantities.

A convenient method for completing this task is to use a mathematical tool called *interval analysis* [15]. Basically interval arithmetics is similar to real arithmetics except that the numbers we are dealing with are intervals. Consequently all the basic operators must be re-defined. For example the addition operator “+” on two intervals $X_1 = [\underline{x}_1, \overline{x}_1]$, $X_2 = [\underline{x}_2, \overline{x}_2]$ is defined as the interval

$$X_1 + X_2 = [\underline{x}_1 + \underline{x}_2, \overline{x}_1 + \overline{x}_2]$$

A nice property of interval arithmetic is that interval operator may be defined for almost any mathematical function. Furthermore if we apply interval analysis on a function it enables us to compute guaranteed lower and upper bounds for the function (that is called an *interval evaluation* of the function), which takes into account rounding errors in the computation. Interval analysis may therefore be used to determine bounds on the quantities defined in the previous section. To calculate the interval evaluation we will use the parser of the ALIAS library¹ that takes as input a file with an analytical description of a function and the ranges for each variable appearing in the function, and returns the interval evaluation of the function. The interval evaluation of a quantity Q will be denoted $\mathcal{B}(Q)$, the lower bound of this interval evaluation $\underline{\mathcal{B}}(Q)$ and its upper bound $\overline{\mathcal{B}}(Q)$. We will also use a *bisection process* on a range $\hat{T} = [T_1, T_2]$: the result of the bisection process applied on this range is the 2 new ranges $[T_1, (T_1 + T_2)/2], [(T_1 + T_2)/2, T_2]$.

4.3 The Algorithm

4.3.1 Principle. We may now describe the principle of our algorithm on an example where we have constraints on the leg lengths, a threshold on the value of the determinant of the inverse jacobian matrix and a validity criteria \mathcal{G} on the passive joints of the robot.

The basic idea of the algorithm is to consider a time interval and to determine if the trajectory is valid for this interval. If we are not able to assert the validity we will use the bisection process to split the interval into two smaller ones that will be processed later on. As the time interval will be considered in sequence it is necessary to be able to store the intervals in a list \mathcal{S} . Associated with this list is an index i which indicates which time interval \mathcal{S}_i , among the n elements of \mathcal{S} , will be processed. The list will be initialized with the range $\mathcal{S}_1 = [0, 1]$ (hence $n = 1$) and the index i will be set to 1. The algorithm proceeds along the following steps:

- 1 if $i > n$ return VALID TRAJECTORY
- 2 compute $\mathcal{B}(\rho_j(\mathcal{S}_i)) j = 1, \dots, 6$:
 - (a) if j exists such that $\mathcal{B}(\rho_j) > \rho_{\max}$ or $\overline{\mathcal{B}}(\rho_j) < \rho_{\min}$, then return INFEASIBLE TRAJECTORY (LEG LENGTHS)
 - (b) if j exists such that $\mathcal{B}(\rho_j) < \rho_{\min}$ or $\overline{\mathcal{B}}(\rho_j) > \rho_{\max}$, then bisect \mathcal{S}_i and add the resulting ranges at the end of \mathcal{S} . Then $i = i + 1$, $n = n + 2$ and go to step 1

¹ALIAS is a C++ package developed in the COPRIN project which enable to analyze and solve system of equations, based on the interval arithmetics package BIAS/Profil

- 3 compute $\mathcal{B}(\mathcal{G})(\mathcal{S}_i)$:
 - (a) if $\overline{\mathcal{B}}(\mathcal{G}) < 0$ then return INFEASIBLE TRAJECTORY (JOINT LIMIT)
 - (b) If $\mathcal{B}(\mathcal{G}) < 0$, then bisect \mathcal{S}_i and add the resulting ranges at the end of \mathcal{S} . Then $i = i + 1$, $n = n + 2$ and go to step 1
- 4 Compute $\mathcal{B}(\mathcal{D})(\mathcal{S}_i)$
 - (a) if $\overline{\mathcal{B}}(\mathcal{D})(\mathcal{S}_i) < \epsilon$, then return INFEASIBLE TRAJECTORY (DEXTERITY)
 - (b) if $\mathcal{B}(\mathcal{D})(\mathcal{S}_i) < \epsilon$ and $\overline{\mathcal{B}}(\mathcal{D})(\mathcal{S}_i) > \epsilon$, then bisect \mathcal{S}_i and add the resulting ranges at the end of \mathcal{S} . Then $i = i + 1$, $n = n + 2$ and go to step 1
- 5 $i = i + 1$ and go to step 1

Consider what will happen with the range \mathcal{S}_1 . At step 2 we will compute the interval evaluation of the 6 leg lengths. At step 2(a) we have found that one of the leg lengths is always lower than ρ_{\min} or greater than ρ_{\max} i.e. the trajectory is outside the workspace. At step 2(b) we have found that the interval evaluation of one of the leg lengths is including the range $[\rho_{\min}, \rho_{\max}]$. But this does not mean that the leg lengths are outside their allowed ranges as interval evaluation may lead to an over-estimation of the bounds. Thus we will bisect the range $\mathcal{S}_1 = [0, 1]$ and start again with the range $\mathcal{S}_2 = [0, 0.5]$ and $\mathcal{S}_3 = [0.5, 1]$. Now assume that the current range \mathcal{S}_i has successfully completed the test 2(a), 2(b) i.e. the leg lengths are all valid. At step 3 we compute the interval evaluation of \mathcal{G} . If the upper bound of this evaluation is negative, then \mathcal{G} will be always negative for any T in \mathcal{S}_i : the trajectory is not feasible (step 3(a)). If the upper bound is positive and the lower bound negative, then we cannot ensure that the trajectory is valid, so we bisect \mathcal{S}_i and start again (step 3(b)). If the current range \mathcal{S}_i has successfully completed test 3(a) and 3(b) we are sure that the trajectory is valid from the view point of the joint limits. Thus we compute an interval evaluation of the dexterity index \mathcal{D} (step 4). If the upper bound of this evaluation is lower than ϵ , then the dexterity for any T in \mathcal{S}_i will always be lower than ϵ i.e. the trajectory is not valid (step 4(a)). If the upper bound is greater than ϵ and the lower bound lower than ϵ , then we cannot ensure that the dexterity constraint is satisfied, so we bisect the box and start again (step 4(b)). If the current time range \mathcal{S}_i has successfully completed the test at step 4, then the part of the trajectory corresponding to \mathcal{S}_i is valid and we proceed with the next range in the list \mathcal{S} (step 5). The algorithm will stop if a part of the trajectory is not valid (steps 2(a), 3(a), 4(a)) or when all the ranges in \mathcal{S} have been processed successfully, which imply that the trajectory is valid (step 1).

4.3.2 Improvement of the Algorithm. Although this algorithm basically looks like the classical “branch-and-bound” method its effective complexity may be poor for complex trajectories. Hence for a practical implementation some methods may have to be used to improve the computation time. Although mathematical details are outside the scope of this paper we will mention some possible ways to improve the efficiency of the algorithm.

A first method is to use the *monotonicity* of the constraints and of the validity criteria. As we have used MAPLE to compute the analytical form of these functions we may also calculate the derivative \mathcal{D}' , ρ'_i , \mathcal{G}' of the equations \mathcal{D} , ρ_i , \mathcal{G} with respect to T . These derivatives may be evaluated using interval analysis and if a derivative has a constant sign, then we will get sharp bounds on the functions. For example if $\mathcal{D}' > 0$ for a given time interval, then \mathcal{D} is monotonically increasing and $\underline{\mathcal{D}}$ will be obtained for the lower bound of the time interval.

Note also that the derivative may be used to perform an interval evaluation of the constraints of the validity criteria using a first order Taylor expansion which, in some cases, may lead to sharper bounds than the natural interval evaluation described in section 4.2 [16]. In the same manner the derivatives may be used to implement an interval Newton method which, under some condi-

tions on the interval evaluation of the derivatives, enable one to determine if there exists a zero of a function $F(T)$ within some range for T . This method may be used for example to test if there is value for T such that on a given trajectory one of the leg lengths is equal to ρ_{\min} or ρ_{\max} .

In the same manner the concept of *consistency* may be used to check a validity criteria [17]. Consistency is a family of methods but we will illustrate one of these methods, the *2B-consistency*, on a simple example: assume that a leg length may be written as $\rho = 2T - T^2 + \sin(T)/3$ and that we want to check if ρ may be larger than 1 when T is in the range $[0, 0.5]$. The interval evaluation of ρ for this interval is $[-0.25, 1.159]$ and hence ρ may indeed be larger than 1. Now we aim to determine if ρ may be equal to 1 for a value of T within the range i.e. to locate a value for T such that $2T - T^2 + \sin(T)/3 = 1$. Using a new variable T_1 this equation may also be written as the system of 2 equations $T_1 = (1 + T^2 - \sin(T)/3)/2$, $T_1 = T$. The interval evaluation of the right side of the first equation allows us to find the bounds $[0.42, 0.625]$ for T_1 and using the second equation we conclude that if ρ is equal to 1, then T must lie within the range $[0, 0.5] \cap [0.42, 0.625] = [0.42, 0.5]$. The interval evaluation of ρ for this new range is $[0.726, 0.983]$ which does not include 1 and hence there is no value of T in $[0, 0.5]$ such that $\rho = 1$. Consequently ρ is either always larger or always smaller than 1: as for $T=0$ we have $\rho = 0$ we deduce that ρ is always smaller than 1 for this time range.

Among the various types of method that may be used to check if a validity criteria is violated let us finally mention the Krawczyk test [18] and Kantorovitch theorem [19] that enable one to determine if a system of equations may have a zero when the unknowns lie within some given ranges.

4.4 Examples and Computation Time. In this section we consider the robot geometry described in the Appendix. In a first example we want to check a circular trajectory for C in the plane $z=56$, the center of the circle being $(0, 0)$ and its radius 3, while the orientation of the platform is fixed with $\psi = \phi = 0$ and $\theta = 5^\circ$. On a SUN Ultra workstation the computation time is 4.50 s: 4.47 s is devoted to the MAPLE calculation and 0.03 s to the trajectory verification itself.

Assume now that the platform center C has still to describe a circular trajectory of center $Q(0,0,56)$ but with the normal of the platform in the plane (QC, z) while maintaining a 5 degree angle with the axis z (this corresponds to a conic motion for the tool). Such trajectory may be defined by:

$$p := 2 * \text{Pi} * T; \quad t := 5 * \text{Pi} / 180; \quad h := 0; \quad x := 3 * \sin(2 * \text{Pi} * T); \\ y := -3 * \cos(2 * \text{Pi} * T); \quad z := 56;$$

After pre-processing the MAPLE file, the algorithm detects in 20 ms that a singularity occurs on the trajectory at some T in the time range $[0.25, 0.375]$. With some minor changes in the algorithm we are then able to detect that a singularity occur at $T=0.250796$. Clearly in that case choosing to maintain the ϕ angle to 0 is a bad choice. If we change the trajectory file to impose $\phi = -\psi$ (this is done by writing $h := -p$: in the trajectory file), the algorithm now find that the trajectory is valid.

Very complex trajectories can easily be checked: consider for example the gear trajectory shown in Fig. 1. The corresponding trajectory file is:

$$p := 0; \quad t := 0; \quad h := 0; \quad z := 56; \quad x := (3 + 0.5 * \sin(40 * \text{Pi} * T)) \\ * \sin(2 * \text{Pi} * T); \\ y := (3 + 0.5 * \sin(40 * \text{Pi} * T)) * \cos(2 * \text{Pi} * T);$$

and the computation time for verifying that this trajectory is valid is 8.24 s.

5 Parametric Trajectories and Uncertainties

5.1 Parametric Trajectories. In the previous examples we have seen that most of the computation time of the algorithm is devoted to the MAPLE computation. But we have a way to decrease this part of the computation time: we will define in the

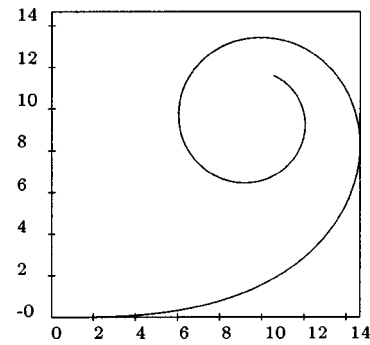


Fig. 3 A clotted

MAPLE trajectory file a *parametric* trajectory. For example the circular trajectory may be defined as:

$$x := x1 * \sin(2 * \text{Pi} * T); \quad y := x2 * \cos(2 * \text{Pi} * T); \quad z := x3; \quad p := x4; \\ h := x5; \quad t := x6;$$

The value of the parameters $x1, x2, x3, x4, x5, x6$ will be defined in an independent file, called the *parameter file*. Then we will run our program once: the analytical forms necessary for our algorithm will include these parameters. After this first run we will indicate to the program that it may re-use the analytical forms established at the first run. We have then only to change the values in the parameter file in order to test any horizontal elliptic or circular trajectory at any orientation. Thus after an initial computation, the computation time for checking the previous circular trajectory lie between 10 to 200 ms on a SUN workstation.

Note that it is possible to indicate in the parameter file successive values for the parameters. Assume, for example that we want to check a horizontal square with corners at $(-1, -1, 56)$ and $(1, 1, 56)$. The trajectory file may be defined as:

$$p := 0; \quad t := 0; \quad h := 0; \quad x := x1 + T * (x2 - x1); \quad y := y1 + T * (y2 - y1); \\ z := 56;$$

which describe a segment trajectory starting at $x1, y1$ and going to $x2, y2$. Then we give in the parameter file the four sets of possible values for these parameters:

$$x1 \ -1 \ x2 \ -1 \ y1 \ -1 \ y2 \ 1 \\ x1 \ -1 \ x2 \ 1 \ y1 \ 1 \ y2 \ 1 \\ x1 \ 1 \ x2 \ -1 \ y1 \ 1 \ y2 \ 1 \\ x1 \ -1 \ x2 \ -1 \ y1 \ 1 \ y2 \ -1$$

and the program will examine in sequence the four segment trajectories in a computation time of 120 ms.

Such a method is quite useful to check trajectories which have not an analytical form. Consider for example the clotted equation:

$$x = \int_0^t 15 \cos(a^2) da \quad y = \int_0^t 15 \sin(a^2) da$$

for t in $[0, 3]$ (Fig. 3). The clotted has been approximated by 250 segments and it takes about 2.9 s to determine that part of the segment from $[14.5582, 7.119]$ to $[14.5743, 7.21]$ is outside the workspace.

5.2 Surface Verification. Up to now we have considered only 1D trajectory but we may easily extend our algorithm to deal with surface verification. A surface is a 2D variety: for example if the platform has to perform a motion such that C lie on sphere centered at x_a, y_a, z_a and radius r while its normal is along the normal of the sphere at C the trajectory file may be written as:

$$x:=x_a-r*\sin(p)*\sin(t);y:=y_a+r*\cos(p)*\sin(t);$$

$$z:=z_a-r*\cos(t);h:=-p;$$

Note that the value of h is arbitrary. Therefore using this model for a sphere we have to deal with two parameters ψ, θ while in the previous sections the only parameter was T . Basically the same “branch and bound algorithm” may be used but its efficiency will be drastically modified by the choice of the heuristics that enable one to improve the interval evaluation of the constraints and validity criteria and determine which variable should be bisected.

As an example we have checked a surface which is a part of a sphere centered at $(0, 0, 59)$ with radius 2, for the range $[0, 2\pi]$ for ψ and $[-0.4, 0.4]$ for θ . The motion verifier finds in 130 ms that the trajectory for ψ in $[0, 0.0982]$ and θ in $[-0.4, -0.3875]$ is not valid.

5.3 Uncertainties in the Trajectories. Up to now we have assumed that the robot will follow exactly the specified trajectory. But in practice control and model errors may cause the real trajectory to be a little bit different from the specified one. To deal with this uncertainty we may assume that each pose parameter τ has an error which may be described by a range $[-\epsilon_\tau, \epsilon_\tau]$. This range is described in an *interval file* as follows:

$$x1 -0.01 0.01$$

$$x2 -0.002 0.002$$

We may then introduce these ranges in the trajectory file. For example the circular trajectory file may be written as:

$$x:=3*\sin(2*Pi*T)+x1; \quad y:=3*\cos(2*Pi*T)+x1;$$

$$z:=56+x1; p:=x2; t:=5*Pi/180+x2; h:=x2;$$

Due to errors in control the trajectory followed by the center of the platform will lie inside a torus and the algorithm checks that any trajectory within this torus is valid. In that case the computation time of the analytical expressions of the constraints and validity criteria is slightly larger (about 1 mn 40 s) but the algorithm determines that the trajectory is still valid.

6 Specific Motion Verifier

Although quite efficient in terms of computation time the generic motion verifier relies on the interpretation mode of the parser to compute the interval evaluation of the validity criterion, an operation that may be used a large number of times for complex trajectories. Clearly a better efficiency would be obtained if these evaluations were done by a devoted C++ module. For a specific parametric trajectory we have developed a MAPLE program that is able to produce automatically the C++ code for all the constraints and validity criteria equations. We are thus able to derive a dedicated motion verifier for a specific type of trajectory. As an example we have developed such a verifier for planar motions of the platform constituted of line segments. Using this specific motion verifier the computation time for the verification of the clotoid trajectory is 150 ms. Thus, a specific motion verifier is at least 10 times faster than the generic verifier.

7 Extension to Other Mechanical Architectures

The previous algorithm may be divided into two parts:

- the MAPLE part in which we compute the analytical form of the criteria we want to check
- the analysis based on these analytical forms

The analysis part is not affected by the mechanical architecture of the robot, while on the other hand the MAPLE part is really

	A_1	A_2	A_3	A_4	A_5	A_6
x	-9	9	12	3	-3	-12
y	9	9	-3	-13	-13	-3
	B_1	B_2	B_3	B_4	B_5	B_6
x	-3	3	7	4	-4	-7
y	7	7	-1	-6	-6	-1

Fig. 4 Coordinates of the A, B points

architecture-dependent. This later part can easily be changed to deal with any mechanical architecture different from the Gough platform, without modifying the analysis part.

8 Conclusion

We have described here a very efficient and user-friendly method for trajectory verification of the motion of parallel structure machines. It enables us to check, for almost any type of trajectories, if the trajectory is fully inside the workspace, is singularity-free and exhibits a good dexterity index, while it can also deal with any other validity criterion as soon as an analytical form is known for the criterion. Although it has been described for the Gough platform it can be easily adapted to any other mechanical architecture.

Prospective work is now to study the problem of the motion planning of parallel robots i.e. to propose an algorithm which will first check if a trajectory is valid, and if not, to propose an alternative valid trajectory

9 Appendix

In the examples we have considered the robot having planar base and platform (hence $z_A = z_B = 0$), described by the coordinates of the A, B points shown in in Fig. 4:

References

- [1] Dasgupta, B., and Mruthyunjaya, T. S., 1998, “Singularity-free Path Planning for the Stewart Platform Manipulator,” *Mech. Mach. Theory*, **33**, No. 6, pp. 711–725.
- [2] Nenchev, D. N., and Uchiyama, M., 1996, “Singularity-consistent Path Planning and Control of Parallel Robot Motion Through Instantaneous-self-motion Type,” In *IEEE Int. Conf. on Robotics and Automation*, pp. 1864–1870.
- [3] Nguyen, C. C. et al., 1992, “Trajectory Planning and Control of a Stewart Platform-based End-effector With Passive Compliance for Part Assembly,” *J. Intell. & Robotic Syst.*, **6**, No. 2-3, pp. 263–281.
- [4] Merlet, J.-P., 1994, “Trajectory Verification in the Workspace for Parallel Manipulators,” *Int. J. Robot. Res.*, **13**, No. 4, pp. 326–333.
- [5] Boudreau, R., and Gosselin, C. M., 1999, “La synthèse d’une plate-forme de Gough-Stewart pour un espace atteignable prescrit,” In *10th World Congress on the Theory of Machines and Mechanisms*, pp. 449–454, Oulu.
- [6] Gosselin, C., Perreault, L., and Vaillancourt, C., 1995, “Simulation and Computer-aided Kinematic Design of three-Degree-of-Freedom Spherical Parallel Manipulators,” *J. Rob. Syst.*, **12**, No. 12, pp. 857–869.
- [7] Ji, Z., 1996, “Analysis of Design Parameters in Platform Manipulators,” *ASME J. Mech. Des.*, **118**, pp. 526–531.
- [8] Merlet, J.-P., 1997, Democrat: “A Design Methodology for the Conception of Robots with Parallel Architecture,” *Robotica*, **15**, No. 4, pp. 367–373.
- [9] Murray, A. P., Pierrot, F., Dauchez, P., and McCarthy, J. M., 1996, “On the Design of Parallel Manipulators for a Prescribed Workspace: A Planar Quaternion Approach,” In J. Lenarčič V. Parenti-Castelli, editor, *Recent Advances in Robot Kinematics*, pp. 349–357, Kluwer.
- [10] Chablat, D., and Wenger, P., 1998, “Moveability and Collision Analysis for Fully-parallel Manipulators,” In *12th RoManSy*, pp. 61–68, Paris.
- [11] Ma, O., and Angeles, J. 1991, “Optimum Architecture Design of Platform Manipulator,” In *ICAR*, pp. 1131–1135.

- [12] Merlet, J-P., 1998, "Determination of the Presence of Singularities in 6D Workspace of a Gough Parallel Manipulator," In *ARK*, pp. 39–48.
- [13] Wenger, P., and Chablat, D., 1998, "Workspace and Assembly Modes in Fully Parallel Manipulators: A Descriptive Study," In *ARK*, pp. 117–126, Strobl.
- [14] Gough, V. E., and Whitehall, S. G., 1962, "Universal Tire Test Machine," In *Proceedings 9th Int. Technical Congress F.I.S.I.T.A.*, **117**, pp. 117–135.
- [15] Yoshikawa, T., 1982, "Manipulability of Robotic Mechanisms," *Int. J. Robot. Res.*, **1**, No. 1.
- [16] Moore, R. E., 1979, "Methods and Applications of Interval Analysis," SIAM Studies in Applied Mathematics.
- [17] Collavizza, M., Deloble, F., and Rueher, M., 1999, "Comparing Partial Consistencies," *Reliable Computing*, **5**, pp. 1–16.
- [18] Krawczyk, R., 1969, "Newton-algorithmen zur Bestimmung von Nullstellen mit Fehlerschranken," *Computing*, **4**, pp. 187–201.
- [19] Tapia, R. A., 1971, "The Kantorovitch Theorem for Newton's Method," *Am. Math. Monthly*, **78**, No. 1ea, pp. 389–392.