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

Abstract

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 verify various criteria which define its validity, for example that the trajectory lie fully inside the workspace of the robot and is singularity-free. We propose an almost real-time method for this purpose 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.

1 Introduction

Off-line verification of a trajectory of a parallel robot is very important in practical application, for example when using such machine for manufacturing operation. Indeed it is necessary to verify if the trajectory is *valid*. i.e. verify a set of criterion, that we will call the *validity criteria*. Example of validity criterion are:

- the trajectory must be 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

The validity criteria may be quite various and so may be the trajectories that have to be checked, from straight lines and arcs of circle to quite complex trajectories as shown in figure 1. Furthermore we will assume that the

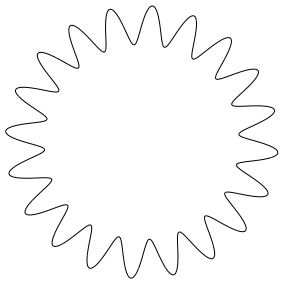

Figure 1: An example of complex trajectory

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 [3] and path planning was addressed from a control view point in [14] and in the neighborhood of a singularity in [13]. An algorithm for checking the validity of a trajectory composed of straight line segments has been proposed in [10] (this paper is an extension of this algorithm).

Another approach is to *design* the robot in such way it verify some validity criteria by design. For example it is possible to determine the robot geometry in such way that its workspace include a specified workspace [1, 4, 6, 8, 12] and then to check that this workspace is singularity-free [2, 7, 9, 15]. But this approach is quite complex and for the time being it can be used for few validity criterion.

Our purpose in this paper is to propose an algorithm which enable to check almost any type of trajectory and set of validity criteria, even very complex one, taking into account the fact that the trajectory followed by the robot will always be a little bit different from the specified one due to control errors.

In this paper we will present our algorithm for the classical Gough-type parallel manipulator [5] illustrated in figure 2, although our algorithm may be used for almost any type of parallel machine. In this robot a base

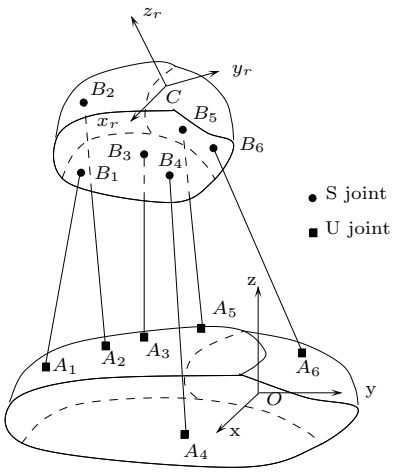

Figure 2: The classical Gough-type parallel robot

and a platform are connected through 6 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 $O, (\mathbf{x}, \mathbf{y}, \mathbf{z})$ which is attached to the base and a mobile frame $C, \mathbf{x_r}, \mathbf{y_r}, \mathbf{z_r})$ 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 time T, supposed to lie in the range [0,1], i.e. that we have

$$x_C = D_x(T) \quad y_C = D_y(T) \quad z_C = D_z(T) \tag{1}$$

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

2.1 Workspace constraints

A first validity criteria can be defined immediately: in practice the leg lengths must lie in some range that will be denoted by $[\rho_{min}, \rho_{max}]$ 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

$$\rho^2 = ||\mathbf{AB}||^2 \qquad \mathbf{AB} = \mathbf{AO} + \mathbf{OC} + \mathbf{CB} \tag{3}$$

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 coordinates vector of B in the mobile frame, which is known.

Using equations (1,2) we may transform equation (3) into a time function. Verifying that ρ lie 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. Consider for example that the ball-and-socket joint at A_1 has a rotation limit of α degree with respect to its main direction defined by the unit vector $\boldsymbol{n_1}$. This imply that we must have

$$\frac{\mathbf{A_1B_1}.\boldsymbol{n_1}}{||\mathbf{A_1B_1}||} > \cos\alpha$$

Using equations (1,2) we may transform this inequality into a time-dependent inequality $\mathcal{G}(T)$ such that for a

valid trajectory $\mathcal{G}(T) > 0$ must be verified for all T in [0,1]. Another interesting validity criteria may be that there is no interference between the legs. If the legs are cylinders it was shown in [10] 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 resolution on the length measurement $\Delta\boldsymbol{\rho}$ by:

$$\Delta\boldsymbol{\rho} = J^{-1}\Delta\mathbf{X} \tag{4}$$

$$\Delta\boldsymbol{X} = J\Delta\boldsymbol{\rho} \tag{5}$$

where J is the jacobian matrix of the robot If we assume that the resolution is identical for all the leg sensors, the leg length measurement errors lie inside an hyper-sphere. This hyper-sphere is mapped through equation (5) into an hyper-ellipsoid. To quantify the shape of the hyper-ellipsoid it is necessary to use a *dexterity index* which may be, for example, the *manipulability* which is $\sqrt{|J^{-1}|}$[16] 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. If κ is 1 the hyper-ellipsoid is a sphere. Both dexterity indices will be 0 at a *singularity* of the robot which is defined by $|J^{-1}| = 0$. Singularities has to be avoided as motion around a singularity will be inaccurate and may lead to very large forces in the legs.

To avoid singularity 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 equation (2) we may transform $\mathcal{D}$ into a time-dependent function $\mathcal{D}(T)$ and a valid trajectory will verify $\mathcal{D}(T) > \epsilon$ for all T in [0,1].

4 The motion verifier

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

$$\rho_{min} \leq \rho_i(T) \leq \rho_{max} \forall i \in [1,6] \tag{6}$$

$$\mathcal{G}_i(T) > 0 \quad \text{for } i \in [1,\ldots,n] \tag{7}$$

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

for all T in [0,1].

4.1 Calculation of the analytical form of the constraints

Our algorithm will make an extensive use of the analytical form of the constraints. 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 (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 we have $\psi = 0$, $\theta = 5$ and $\phi = 0$ degree.

Another example is the gear trajectory shown in figure 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 MAPLE programs, called the *constraint file* which will enable to calculate the analytical form of the constraints $\mathcal{G}_i$. As soon as the `data` file, the trajectory file `trajectory` and the constraint file have been defined the program will run the following specific MAPLE program, called `main`:

```
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 expression 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 Annex the analytical form of `RO1` is:

```
3311+784*sin(Pi/36)-126*cos(Pi/36)+(42*cos(Pi/36)-54)*cos(2*Pi*T)+36*sin(2*Pi*T)
```

4.2 Interval analysis

Our algorithm will require to compute 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* [11]. Basically interval arithmetics is similar to real arithmetics except that the number 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 arithmetics is that interval operator may be defined for almost any mathematical functions. Furthermore if we apply interval analysis on a function it enable to compute guaranteed lower and upper bounds for the function (that is called an *interval evaluation* of the function), including even rounding errors in the computation. Interval analysis may therefore be used to determine bounds on the quantities defined in the previous section. To perform the interval evaluation we will use the parser of the `ALIAS` library[1] that take 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* for the 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 constraint on the leg lengths, a threshold on the value of the determinant of the inverse jacobian matrix and a constraint $\mathcal{G}$ on a passive joint of the robot.

We will use a list $\mathcal{S}$ of ranges for T with n ranges. This list will be initialized with the range $S_1 = [0, 1]$ and hence $n = 1$. During the algorithm we will consider the i-th element $\mathcal{S}_i$ of the list $\mathcal{S}$. We start with $i = 1$ and the algorithm proceeds along the following steps:

1. if $i > n$ return `VALID TRAJECTORY`

[1] `ALIAS` is a package developed in the `SAGA` project which enable to analyze and solve system of equations, based on the interval arithmetics package BIAS/Profil

2. compute $\mathcal{B}(\rho_i(\mathcal{S}_i))$:

   (a) if it exists i such that $\underline{\mathcal{B}(\rho_i)} > \rho_{max}$ or $\overline{\mathcal{B}(\rho_i)} < \rho_{min}$, then return `INFEASIBLE TRAJECTORY (LEG LENGTHS)`

   (b) if it exists i such that $\underline{\mathcal{B}(\rho_i)} < \rho_{min}$ or $\overline{\mathcal{B}(\rho_i)} > \rho_{max}$, 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

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 $\underline{\mathcal{B}(\mathcal{G})} < 0$, 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 $\underline{\mathcal{B}(\mathcal{D})}(\mathcal{S}_i) < \epsilon$ 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 length include the range $[\rho_{min}, \rho_{max}]$. But this does not mean that the leg length 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 feasible, so we bisect $\mathcal{S}_i$ and start again (step 3(b)). If the current range $\mathcal{S}_i$ has successfully completed the test 3(a) and 3(b) we are sure that the trajectory is feasible 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 infeasible (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

The practical implementation of the previous algorithm includes in fact numerous tricks to improve the computation time. The first trick is to memorize information when bisecting. Assume for example that the range $\mathcal{S}_{99}$ and $\mathcal{S}_{100}$ result from the bisection of the range $\mathcal{S}_{10}$ for which the leg lengths ρ_1 satisfy the constraint $\rho_{min} \leq \rho_1 \leq \rho_{max}$. Consequently there is no need to calculate the interval evaluation of ρ_1 for $\mathcal{S}_{99}$ and $\mathcal{S}_{100}$ as we are sure that it will satisfy the constraint.

A second trick is to manage more cleverly the list $\mathcal{S}$. When bisecting the range $\mathcal{S}_i$ instead of adding the two new ranges at the end of $\mathcal{S}$ we may substitute the current range $\mathcal{S}_i$ by one of the two ranges resulting from the bisection, while storing the other range at the end of the list (which imply that we do not increment i). Now we have to determine which of the two ranges will be chosen to replace $\mathcal{S}_i$. Assume for example that the two ranges $\mathcal{S}^1, \mathcal{S}^2$ resulting from the bisection of $\mathcal{S}_i$ must be stored because their lower bounds s^1, s^2 on $\mathcal{D}$ are lower than ϵ. A good choice will be to substitute $\mathcal{S}_i$ by the range having the largest ratio $|s^k - \epsilon|/\epsilon$. Indeed if the corresponding range includes a range for which $\mathcal{D}$ is lower than ϵ it is probable that by using this method the

range will be on top of the list quite quickly.

A third trick is to use the *monotonicity* of the constraint. As we have used MAPLE to compute the analytical form of the constraint we may also calculate the derivative $\mathcal{D}^T, \rho_i^T, \mathcal{G}^T$ of the constraint equations $\mathcal{D}, \rho_i, \mathcal{G}$ 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 constraint. For example if $\underline{\mathcal{D}^T} > 0$ we will have $\underline{\mathcal{D}} = \mathcal{D}(0)$ and if $\mathcal{D}(0) < \epsilon$, then we are sure that the current range $\mathcal{S}_i$ includes a range for which $\mathcal{D} < \epsilon$ and the trajectory is not valid.

4.4 Examples and computation time

In this section we consider the robot described in the Annex. In a first example we want to verify a circular trajectory in the plane $z = 56$, the center of the circle being (0,0) an its radius 3, with $\psi = \phi = 0$ and $\theta = 5°$. On a SUN Ultra1 workstation the computation time is 4.50s: 4.47s is devoted to the MAPLE calculation and 0.03s to the trajectory verification itself.

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

```
p:=2*Pi*T: t:=5*Pi/180: h:=0: x:=3*sin(2*Pi*T): y:=-3*cos(2*Pi*T): z:=56:
```

After pre-processing the MAPLE file, the algorithm detect in 20ms that a singularity occur 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 figure 1. The corresponding 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):
```

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

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 the computation time: we will define in the MAPLE trajectory file a *parametric* trajectory. For example the circular trajectory may be defined as:

```
x:=x1*sin(2*Pi*T):y:=x2*cos(2*Pi*T): z:=x3: p:=x4: h:=x5: 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 previous calculation on the circular trajectory for a radius of 13.549 is completed in 180ms (the trajectory is still valid), while for a radius of 13.55 the algorithm determines that for T in [0.304688,0.3125] the trajectory is outside the workspace in a computation time of 60 ms.

5.2 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 induce that the real trajectory is a little bit different from the specified one. To deal with this uncertainty we may assume that each pose parameter r has an error which may be described by a range $[-\epsilon_r, \epsilon_r]$. This range are described in an *interval file* as follow:

```
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: y:=3*cos(2*Pi*T)+x1:

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

The computation of the analytical form is slightly larger (about 1mn40s) but the algorithm determine that the trajectory is still valid.

6 Set of trajectories

Our program is also able to check a set of trajectories. Assume, for example that you want to check a horizontal square with corners at (-1,-1,56) and (1,1,56). Thus you have to test a set of segment trajectory. The trajectory file may be defined as:

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

which describe a segment trajectory starting at `x1, y1` and going to `x2, y2`. Then you 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 120ms.

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

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

for t in [0,3] (figure 3). The clotoid has been approximated by 250 segments and it takes about 2.9s to determine

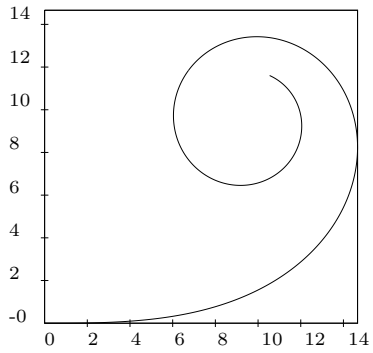

Figure 3: A clotoid

that part of the segment from [14.5582,7.119] to [14.5743,7.21] is outside the workspace.

7 Specific motion verifier

Although quite efficient in term of computation time the generic motion verifier rely on the parser to compute the interval evaluation of the validity criterion. Clearly a better efficiency will be obtained if these evaluations were done by a devoted C++ module. For a specific parametric trajectory we have developed a MAPLE program based on the `main` program that is able to produce automatically the C++ code for all the constraint equations. We are thus able to derive a specific motion verifier for a specific type of trajectory. As an example we have developed such verifier for planar motions of the platform constituted of line segment. Using this specific motion verifier the computation time for the verification of the clotoid trajectory is 150ms. Thus, a specific motion verifier is at least 10 time faster than the generic verifier.

8 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 `xa, ya, za` and radius `r` while its normal is along the normal of the sphere at C the trajectory file may be written as:

```
x:=xa-r*sin(p)*sin(t):y:=ya+r*cos(p)*sin(t):z:=za-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 section the only parameter was T. But basically the same algorithm may be used except that the bisection process will lead to 4 new sets of ranges. Otherwise the algorithm remains the same. 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 find in 130ms that the trajectory for ψ in [0,0.0982] and θ in [-0.4,-0.3875] is not valid.

9 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 really affected by the mechanical architecture of the robot, while on the other hand the MAPLE part is really architecture-dependent. But this part is composed of a MAPLE program, `main`, which is external to the main program. Thus, just by modifying `main` to adapt it to a mechanical architecture different from the one of the Gough platform we may easily modify our program to deal with almost any mechanical architecture.

10 Conclusion

We have described here a very efficient and user-friendly method for trajectory verification of the motion of parallel structure machine. It enable to check for almost any type of trajectories if the trajectory is in the workspace, is singularity-free and exhibit a good dexterity index, while it can also deal with any other validity criteria as soon as an analytical form is known for the criteria. Similarly, 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 robot i.e. to propose an algorithm to check if a trajectory is feasible, and if not, to propose correction on the specified trajectory so that the corrected trajectory is feasible.

11 Annex

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

	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

References

[1] Boudreau R. and Gosselin C.M. 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*, pages 449–454, Oulu, June, 20-24, 1999.

[2] Chablat D. and Wenger P. Moveability and collision analysis for fully-parallel manipulators. In *12th RoManSy*, pages 61–68, Paris, July, 6-9, 1998.

[3] Dasgupta B. and Mruthyunjaya T.S. Singularity-free path planning for the Stewart platform manipulator. *Mechanism and Machine Theory*, 33(6):711–725, August 1998.

[4] Gosselin C., Perreault L., and Vaillancourt C. Simulation and computer-aided kinematic design of three-degree-of-freedom spherical parallel manipulators. *J. of Robotic Systems*, 12(12):857–869, 1995.

[5] Gough V.E. and Whitehall S.G. Universal tire test machine. In *Proceedings 9th Int. Technical Congress F.I.S.I.T.A.*, volume 117, pages 117–135, May 1962.

[6] Ji Z. Analysis of design parameters in platform manipulators. *ASME J. of Mechanical Design*, 118:526–531, December 1996.

[7] Ma O. and Angeles J. Optimum architecture design of platform manipulator. In *ICAR*, pages 1131–1135, Pise, June, 19-22, 1991.

[8] Merlet J-P. Democrat: A DEsign Methodology for the Conception of robots with parallel ArchiTecture. *Robotica*, 15(4):367–373, July - August , 1997.

[9] Merlet J-P. Determination of the presence of singularities in 6D workspace of a Gough parallel manipulator. In *ARK*, pages 39–48, Strobl, June 29- July 4, 1998.

[10] Merlet J-P. Trajectory verification in the workspace for parallel manipulators. *Int. J. of Robotics Research*, 13(4):326–333, August 1994.

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

[12] Murray A.P., Pierrot F., Dauchez P., and McCarthy J.M. 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*, pages 349–357. Kluwer, 1996.

[13] Nenchev D.N. and Uchiyama M. Singularity-consistent path planning and control of parallel robot motion through instantaneous-self-motion type. In *IEEE Int. Conf. on Robotics and Automation*, pages 1864–1870, Minneapolis, April, 24-26, 1996.

[14] Nguyen C.C. and others . Trajectory planning and control of a Stewart platform-based end-effector with passive compliance for part assembly. *J. of Intelligent and Robotics Systems*, 6(2-3):263–281, December 1992.

[15] Wenger P. and Chablat D. Workspace and assembly modes in fully parallel manipulators: a descriptive study. In *ARK*, pages 117–126, Strobl, June 29- July 4, 1998.

[16] Yoshikawa T. Manipulability of robotic mechanisms. *The Int. J. of Robotics Research*, 1(1), 1982.