UCLouvain

École polytechnique de Louvain

Analysis and Implementation of Redundant Constraints in Multibody Systems

Author: **Arthur Paul**
Supervisor: **Paul Fisette**
Readers: **Nicolas Docquier, Vincent Legat**
Academic year 2019–2020
Master [120] in Electro-mechanical Engineering

Contents

1 Introduction

In the field of multibody system dynamics, a topic of interest is the occurrence of redundant constraints in mechanisms. Those are the constraints that, if removed, would not change the kinematics of the system. However, detecting and treating such constraints brings about a load of challenge.

We encounter mechanisms containing redundant constraints everywhere and they are not simply decorative. As an example, a simple door comprises many hinges that restrain the same degree of freedom, it is required as using only one hinge, when possible, could lead to the failure of said door, the many hinges are here in order to avoid failure by dividing the effort between these hinges. They can also be in used in more active processes, the agile eye (that will be simulated in this thesis) being one of them.

The goal of this thesis is firstly to understand what these constraints are and what their effect is on the simulation of a multibody system, and secondly to implement them in the software developed by the Université Catholique de Louvain, Robotran. Robotran is a software used to model and analyse multibody systems and is used both in the research and industrial field. More precisely, we will use the matlab version of Robotran.

This thesis will be divided into different parts. Firstly, the state of the art will be looked into. It will be followed by explanations on the different elements and notations of a multibody system as well as the mechanisms involved in the simulation of a multibody system, it will mainly be focused on what is important for this thesis and some features that do not exert any influence on redundant constraints will be ignored. A statement of the kind of problem we want to resolve will shortly be made and then, the methodology used to implement redundant constraints in Robotran will be thoroughly detailed. Finally, we will put this implementation to the test by simulating mechanisms containing redundant constraints.

2 State of the art

Researches linked to redundant constraints can be found in many fields. While it is more commonly talked about in applied mathematics or economy, valuable breakthroughs have also been made in the field of multibody systems. The scientific review "Multibody System Dynamics" published by Springer provides a reliable source of knowledge concerning multibody systems, and it happens with no surprise that a few articles treat the case of the redundant constraints in mechanisms.

In 2003, an article [1] was released by Maria Augusta Neto and Jorge Ambrósio, it proposes methods in order to integrate DAE (form by which a multibody system is characterised) in the presence of redundant constraints: The Baumgarte stabilisation method, the augmented Lagrangian method and the coordinate partitioning method. One of them, the coordinate partitioning (used by Robotran) is of particular interest to us: this method, as we will see in this thesis, has the advantage to detect, and thus to eliminate, redundant constraints in the factorisation phase of the process. Although this article already gives a large amount of knowledge concerning the resolution of dynamic equations, it does not entirely provide everything that is needed in order to tackle the potential issue that can arise with redundant constraints.

In 2005, an article [2] was released by Marek Wojtyra. It proposed ways to compute the joint reaction forces in multibody systems containing redundant constraints. He stated quite correctly that the choice of which constraint was redundant was arbitrary, as "if the role of constraint A can be played by the constraint B, then the role of constraint B can also be played by constraint A". The issue addressed in his article was to be able to determine uniquely all constraint reactions in a multibody system. The method he presented enabled the detection of constraints and joints for which reactions can be uniquely determined despite the presence of redundant constraints, the latter not being uniquely identifiable.

In 2013, an article [3] was released by Javier García de Jalón. In his paper, he focused on formulation methods for determining the equations of motion of multibody systems, considering there are redundant constraints.

In a way, this thesis will contribute to somewhat complete what has already been done regarding the detection and implementation of redundant constraints in multibody systems.

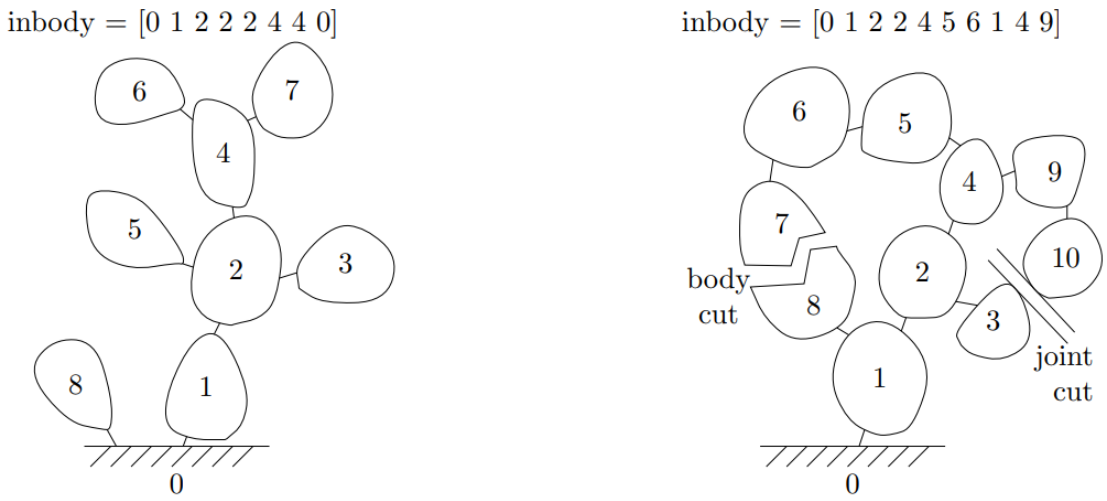
3 Multibody systems

This part is a summary of the Robotran manual [4] and is thus highly similar.

3.1 Representation of a multibody system

3.1.1 Topology

A general multibody system is a mechanical system composed of n rigid bodies, these bodies are connected by joints and form tree-like structures with no loops as shown in Fig.1(a). In case of the mechanical system containing loops (e.g. car suspensions), either a body or a joint will be cut to restore a tree-like structure (see Fig.1(b))The joint cut will impose 3 constraints in order for two points to be at the same location, while the solid cut will impose 6 constraints to the original body and "phantom" body (in the image: body 7 and body 8) in order for the two bodies' frames to have the same location and orientation.The topological vector *inbody* is defined as the vector whose i^{th} element contains the index of the parent of body i.



(a) Tree-like structure

(b) Closed-loop structure

Figure 1: Topology of a multibody system. Figure from [4]

3.1.2 Bodies

As shown in fig.2, an orthogonal right-handed body frame $\{\hat{\mathbf{X}}^i\}$ is rigidly fixed to each body i and is located at its center of mass. Before going further, we will

define a general vector $\mathbf{a}$ as well as a general tensor $\mathbf{T}$ in a given body frame $\{\hat{\mathbf{X}}^i\}$. In general, the vector $\mathbf{a}$ is written:

$$\mathbf{a} = a_x\hat{\mathbf{X}}^i_x + a_y\hat{\mathbf{X}}^i_y + a_z\hat{\mathbf{X}}^i_z \tag{1}$$

We can make this expression more compact by introducing the following notations:

$$a \triangleq \begin{pmatrix} a_x & a_y & a_z \end{pmatrix}^t \tag{2}$$

$$\left[\hat{\mathbf{X}}^i\right] \triangleq \begin{pmatrix} \hat{\mathbf{X}}^i_x \\ \hat{\mathbf{X}}^i_y \\ \hat{\mathbf{X}}^i_z \end{pmatrix} \tag{3}$$

Which gives us:

$$\mathbf{a} = \left[\hat{\mathbf{X}}^i\right]^t a \tag{4}$$

The same goes for a tensor $\mathbf{T}$, by defining:

$$T = \begin{pmatrix} T_{xx} & T_{xy} & T_{xz} \\ T_{yx} & T_{yy} & T_{yz} \\ T_{zx} & T_{zy} & T_{zz} \end{pmatrix} \tag{5}$$

We get:

$$\mathbf{T} = \left[\hat{\mathbf{X}}^i\right]^t T \left[\hat{\mathbf{X}}^i\right] \tag{6}$$

Now that the tensors and vectors have been defined, we can analyse the different components of a body and its neighbouring bodies (see Fig.2).

- By convention, a joint's index is the same as its succeeding body and two reference anchor points are defined: O^i on the parent body of body i and O'^i on body i itself.

- The joint vector locating the connection point O^j of joint j with respect to the connection point O'^i of joint i on body i is introduced by the vector $\mathbf{d}^j = \left[\hat{\mathbf{X}}^i\right]^t d^j$.

- The vector locating the center of mass CM^i of body i with respect to point O'^i is introduced by the vector $\mathbf{l}^j = \left[\hat{\mathbf{X}}^i\right]^t l^j$.

- The mass and inertia tensor (with respect to its center of mass) of body i are introduced by m^i and $\mathbf{I}^i = \left[\hat{\mathbf{X}}^i\right]^t I^i \left[\hat{\mathbf{X}}^i\right]$

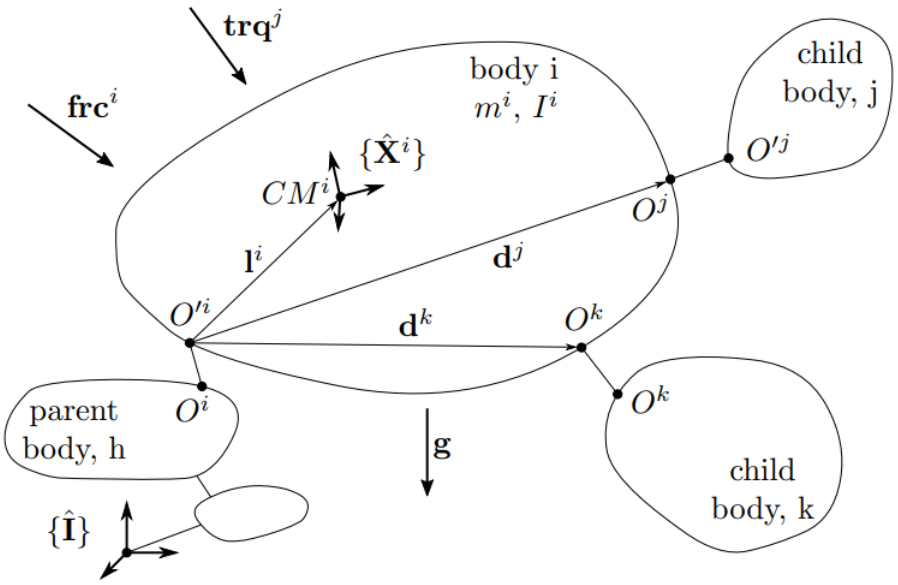


Figure 2: Body representation in a multibody system. Figure from [4]

3.1.3 Joints

Joints between bodies represent any kind of device that induce motion between two bodies (e.g. hinges, telescopic arms, ball joints etc., see Fig.3)

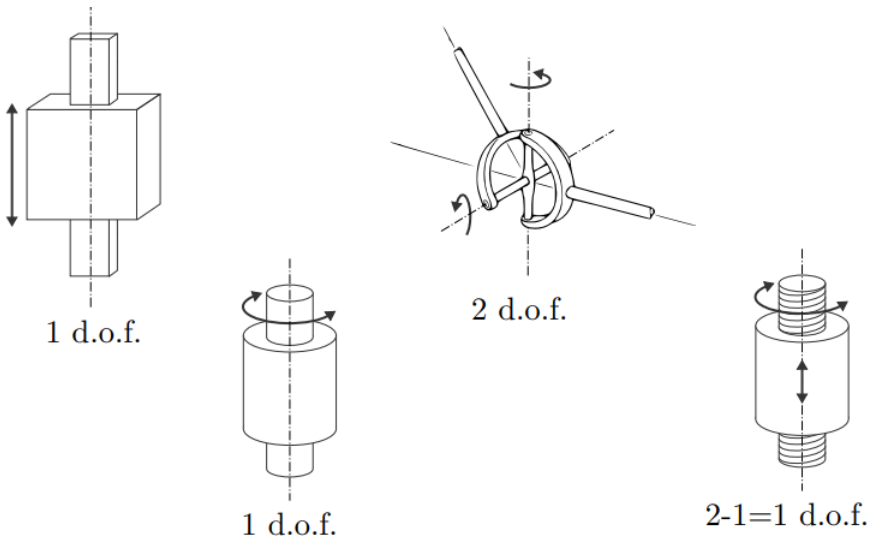


Figure 3: Examples of different joints. Figure from [4]

In Robotran, relative motions in these joints are described by the general coordinates q and their time derivatives $\dot{q}$ and $\ddot{q}$. Furthermore, in order not to have to build a library of joints and to keep it simple, joints in Robotran can only have one degree of freedom, a prismatic one (denoted T) and a revolute one (denoted R). By introducing the joint unit vector $\hat{\mathbf{e}}^j$, aligned with the joint axis, we can express what the generalised coordinate q^j in joint j represents:

- If joint j is prismatic, q^j represents a relative displacement (in $[m]$) such that $\overrightarrow{O^j O'^j} = q^j \hat{\mathbf{e}}^j$ (see Fig.4(a)).

- If joint j is revolute, q^j represents a relative angle (in $[rad]$) which characterises a rotation along $\hat{\mathbf{e}}^j$ of body frame $\left\{\hat{\mathbf{X}}^j\right\}$ with respect to parent body frame $\left\{\hat{\mathbf{X}}^i\right\}$ (see Fig.4(b)). In Robotran, in order to avoid multiple frames per body, a specific convention is used: the axis of a joint must be aligned with one of the three axis direction of the body frame.

(a) Prismatic joint (b) Revolute joint

Figure 4: Elementary joints. Figure from [4]

Regarding more complex joints, they can be modelled as a succession of elementary joints and so called fictitious bodies. An example of the modelling of such a joint (2 d.o.f. universal joint) is shown in Fig.5.

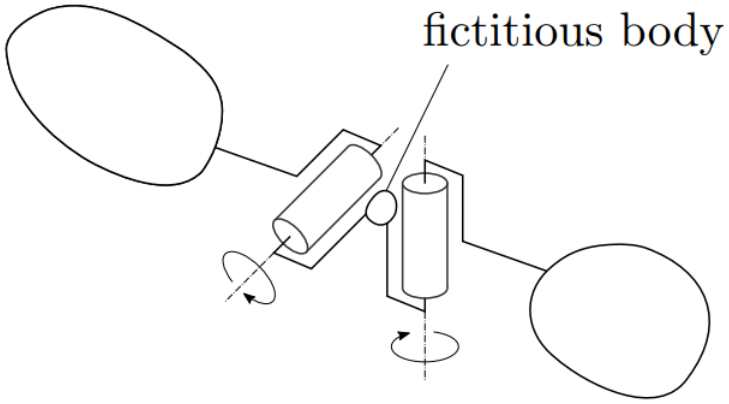


Figure 5: modelling of a 2 d.o.f. universal joint. Figure from [4]

Although the Robotran specific hypothesis related to the joint direction (the one that defines the axis of a joint as being systematically aligned with one of the axis direction of the body frame) seems quite restrictive, it is barely the case: a skewed joint in the real system can be modelled by introducing intermediary joints (and massless fictitious bodies) that will be locked to a constant value during the simulation. In Robotran, such joints are called **Driven/Forced joint**. An example of such a situation is shown in Fig.6.

(a) real system

(b) model

Figure 6: modelling of a skewed joint. Figure from [4]

3.2 Multibody formalisms

3.2.1 Constitutive equations

The equations describing the dynamics of an unconstrained system of n bodies, a.k.a. the equations of motion, can be written as follows:

$$M(q)\ddot{q} + c(q, \dot{q}, g) = Q(q, \dot{q}) \tag{7}$$

Where:

- M [n * n] is the mass matrix
- c [n * 1] is the nonlinear dynamic vector which contains the gyroscopic, centrifugal and gravity terms.
- Q [n * 1] represents the generalised joint forces/torques

As aforementioned, it is quite common to encounter multibody systems in which there are kinematic loops. In order to model them and still have tree-like structures, constraints must be introduced to close these loops. The m constraints are written in terms of the generalised coordinates q and will be denoted as $h(q)$,it will be of dimension $[m * 1]$. In order to close these loops, $h(q) = 0$ must be verified.
In order to describe the complete system, the constraints and equations of motion must be added. By using Lagrange's multipliers technique to introduce the constraint forces, the system becomes:

$$M(q)\ddot{q} + c(q, \dot{q}, g) = Q(q, \dot{q}) + J^T\lambda \tag{8}$$
$$h(q) = 0 \tag{9}$$
$$\dot{h}(q, \dot{q}) = J(q)\dot{q} = 0 \tag{10}$$
$$\ddot{h}(q, \dot{q}, \ddot{q}) = J(q)\ddot{q} + \dot{J}\dot{q}(q, \dot{q}) \tag{11}$$

3.2.2 Solving method

Many methods can be used in order to solve this system of differential algebraic equations (DAE). In this case, a method to perform a full index reduction of the system was chosen in order to have purely differential form. This method is called the coordinate partitioning method.

In order for this method to be valid, the Jacobian matrix $J(q)$ must be regular (i.e. full rank). This implies that the constraints $h(q) = 0$ are independent from each other. The principle of this method is to partition the n generalised coordinates into two groups: one of size m that can be locally expressed as function of the other one of size $(n - m)$. It allows the system to be reduced to a set of $(n - m)$ differential equations.
First, the coordinate partitioning is performed, the vector of generalised coordinates is reorganised into two subsets, q_u of size $(n - m)$ and q_v of size m. By performing the adequate re-orderings of the columns, the Jacobian J is also partitioned into two subsets, J_u and J_v. in other words:

$$q = \begin{pmatrix} q_u \\ q_v \end{pmatrix} \tag{12}$$
$$J = \begin{pmatrix} J_u & J_v \end{pmatrix} \tag{13}$$

The equation eq.(8) can be rewritten:

$$\begin{pmatrix} M_{uu} & M_{uv} \\ M_{vu} & M_{vv} \end{pmatrix} \begin{pmatrix} \ddot{q}_u \\ \ddot{q}_v \end{pmatrix} + \begin{pmatrix} c_u \\ c_v \end{pmatrix} = \begin{pmatrix} Q_u \\ Q_v \end{pmatrix} + \begin{pmatrix} J_u^t \\ J_v^t \end{pmatrix} \lambda \tag{14}$$

Since J_v is assumed to be regular (independent constraints), it is then possible to eliminate the unknowns λ using the lower part of the system:

$$\begin{pmatrix} M_{uu} & M_{uv} \end{pmatrix} \begin{pmatrix} \ddot{q}_u \\ \ddot{q}_v \end{pmatrix} + B_{vu}^t \begin{pmatrix} M_{uu} & M_{uv} \end{pmatrix} \begin{pmatrix} \ddot{q}_u \\ \ddot{q}_v \end{pmatrix} + c_u + B_{vu}^t c_v = Q_u + B_{vu}^t Q_v \tag{15}$$

where $B_{vu} \triangleq -J_v^{-1} J_u$ is defined as the coupling matrix. The next step is to eliminate the dependent variable q_v by solving the constraints. Due to the possibly

non-linear nature of these constraints, analytical solutions are not enough and an iterative procedure is required. The Newton-Raphson algorithm can be used to find q_v for a given q_u through successive estimations of q_v. In this case, this method gives:

$$q_v^{k+1} = q_v^k - (J_v)^{-1} h|_{q_v=q_v^k} \tag{16}$$

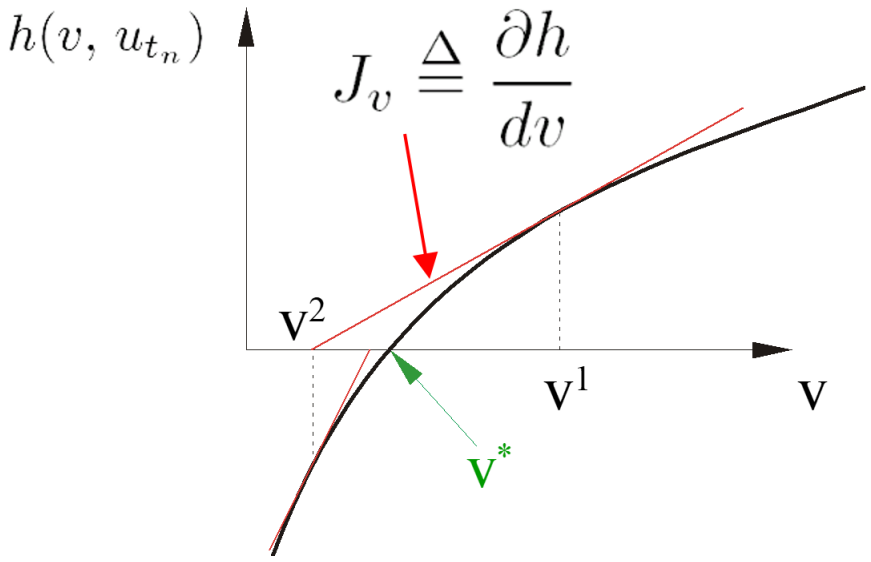


Figure 7: Newton-Raphson method. Figure from [5]

In turn,$\dot{q}_v$ and $\ddot{q}_v$ can be determined using eq.(10) and eq.(11):

$$\dot{q}_v = B_{vu}\dot{q}_u \tag{17}$$
$$\ddot{q}_v = B_{vu}\ddot{q}_u + b \tag{18}$$
$$b \triangleq -J_v^{-1}(\dot{J}\dot{q}) \tag{19}$$

The final reduced system becomes:

$$(M_{uu}+M_{uv}B_{vu}+B_{vu}^t M_{vu}+B_{vu}^t M_{vv}B_{vu})\ddot{q}_u+(M_{uv}B_{vu}^t M_{vv})b+(c_u+B_{vu}^t c_v)-(Q_u+B_{vu}^t Q_v) = 0 \tag{20}$$

Or more concisely:

$$M_r(q_u)\ddot{q}_u + F_r(\dot{q}_u, q_u) \tag{21}$$

This set of differential equations constitutes the equations of motion of the constrained system described in terms of the $(n-m)$ independent generalized coordinates q_u. A summary of the workflow of the coordinate partitioning is shown in Fig.8.

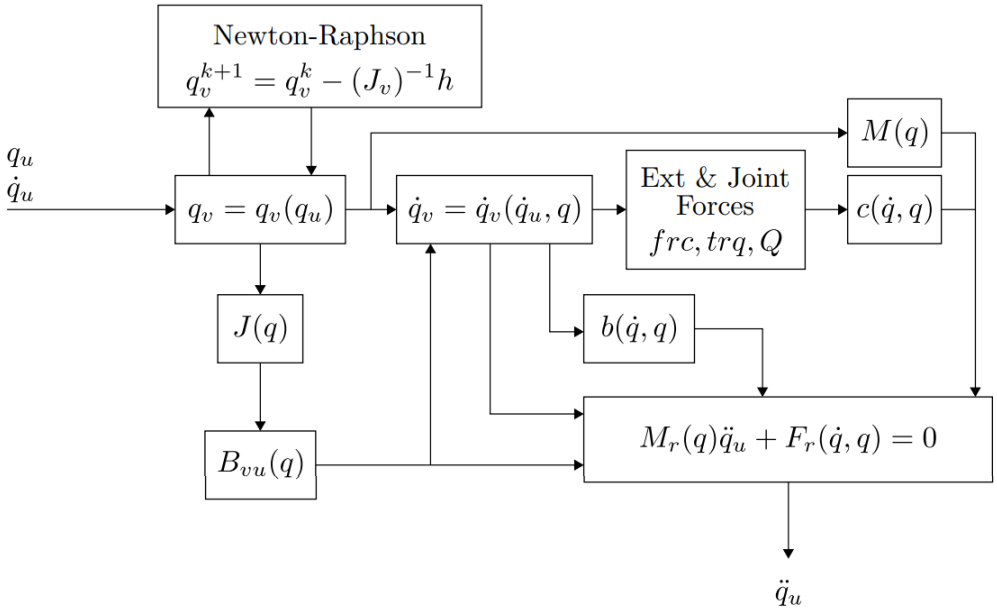


Figure 8: Workflow chart of the coordinate partitioning. Figure from [4]

3.2.3 Coordinate partitioning

An important question to ask is how to choose which generalised coordinates are dependent or independent.
First, it is important to note that the number of independent coordinates must match the number of degrees of freedom of the system. Then, by looking at the definition of B_{vu}, eq.(16) and Fig.7, one can see that the dependent generalised coordinates q_v should be chosen such as to have the best conditioned sub-Jacobian J_v as possible, as it will make the system more robust. While it can be achieved "manually" for smaller multibody systems, it becomes increasingly difficult the bigger this system gets.
A good means to obtain a well conditioned J_v is to perform an LU factorisation [6] of the whole rectangular Jacobian matrix $J(q)$. In our case, this factorisation aims to place the largest possible pivot (in absolute value) on the factorised matrix diagonal by permuting the rows and columns of the matrix.

The working principle of this factorisation is illustrated in Fig.9; assuming the largest element of the Jacobian is the element 1, it is "moved" to the first element of the diagonal of the left side submatrix (in our case J_v). The needed permutations are performed and recorded, and the same process happens again for the second element of the left submatrix, then again for the third one, etc. It is also important to point that the search of the biggest pivot does not always take place in the whole

matrix, the search domain in a matrix $M(1:m, 1:n)$ for the element i of the left submatrix diagonal is the submatrix $M(i:m, i:n)$. The process ends when all the rows of the matrix have been processed or when a pivot is equal to zero.

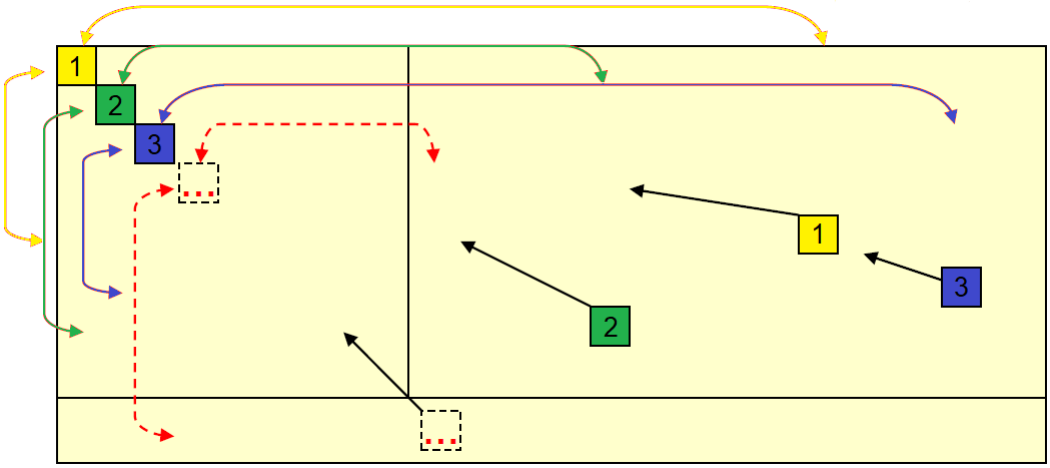


Figure 9: Principle of the LU factorisation with full pivoting. Figure adapted from [5]

As aforementioned, it is possible that a pivot remains very small (≈ 0). In this case, it means that a constraint was not independent from the others, the Jacobian is not full rank, in other words, there is a **redundant constraint**. The advantage of this factorisation method is its ability to detect these redundant constraints, we can thus weed them out of our system. The final state of the Jacobian should have the form shown in Fig.10

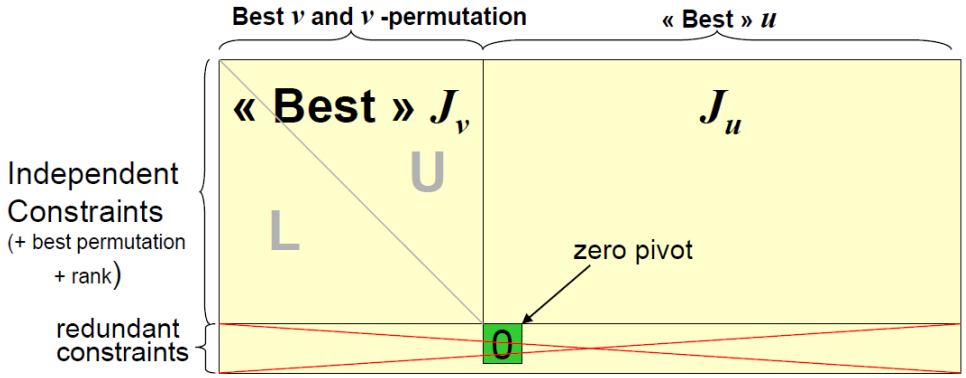


Figure 10: LU factorisation with full pivoting with redundant constraints. Figure from [5]

In Robotran, the choice of independent coordinates can be totally or partially set by the user. In order to preserve the user's choices, some of the columns can be locked. This principle is shown in Fig.11.

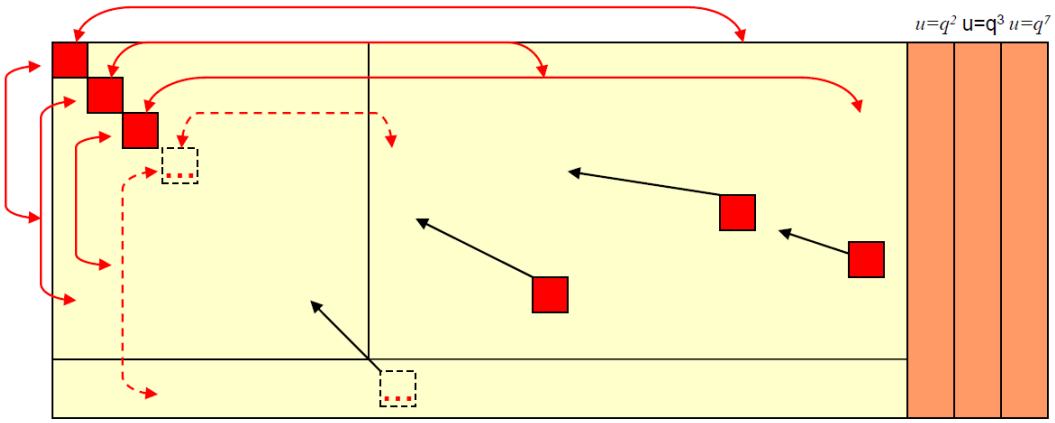


Figure 11: Principle of the LU factorisation with full pivoting and some locked independent variables. Figure from [5]

4 Identification of the problem

4.1 The redundant constraints

As its name implies, a redundant constraint is a constraint that has already been expressed and is thus not needed. To understand better the problems caused by redundant constraints in the multibody field (as well as the redundant constraints themselves), an analysis of the degrees of freedom of a structure containing redundancies must be performed. An example of such a structure is the four bars system shown in Fig.12. This structure is purely two-dimensional and has an arm that does not provide any kinematic change in the structure (redundant). Each arm has 3 d.o.f., and each joint induces two translation constraints. The total number of degrees of freedom of a system is the number of individual degrees of freedom minus the number of constraints. In this case, it would mean 0 degrees of freedom. However, it is obvious that the system has in reality 1 degree of freedom, it means that one of the constraints is redundant. In fact, in one of the joints, one constraint of translation coupled with the constraints in the other joints systematically implies the other constraint of translation in the joint.

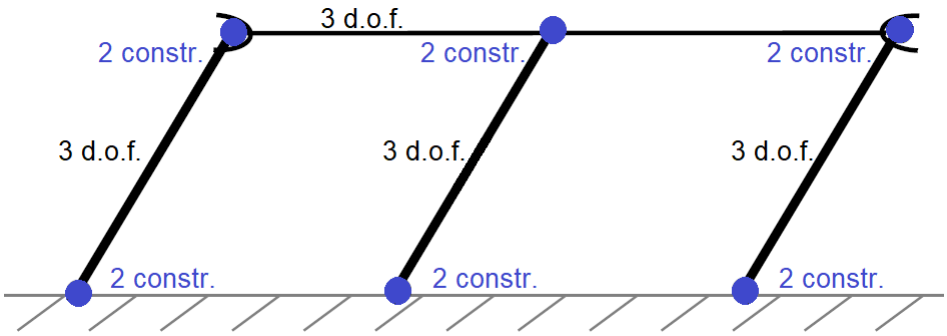


Figure 12: Redundant structure. Figure adapted from[5]

It is relatively easy to find the redundant constraints in such small structures, it becomes much more difficult for bigger systems. A good example for a complex structure containing redundancies would be the agile eye [7], a mechanical system used to achieve the fast orientation of a camera (see Fig.13). Although a classical analysis of this system would give -3 d.o.f., this system has three d.o.f., which implies that there are 6 redundant constraints.

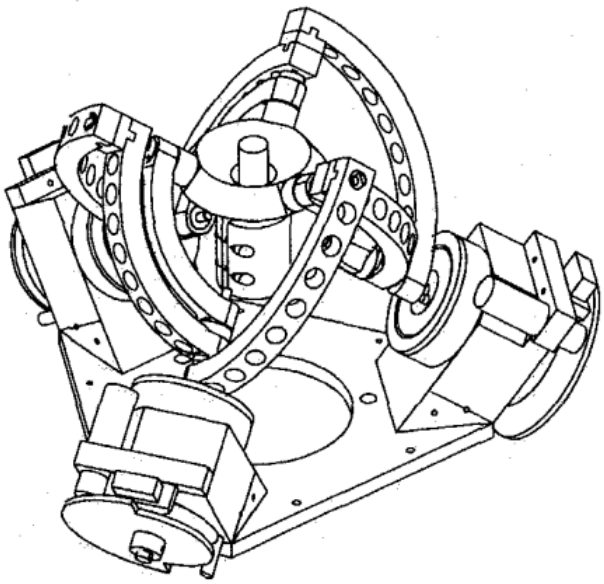


Figure 13: Agile eye. Figure from [8]

4.2 Main issue

The coordinate partitioning mentioned in section 3.2.3 may seem fine, but a problem remains. In order to detect a redundant constraint in the Jacobian and to get the best possible J_v by using the LU factorisation, the system must be closed, i.e. the constraints must be satisfied, which is rarely the case in practical multibody modelling (the modelled structures are not systematically represented in their closed state). However, in order to close the loops using the Newton-Raphson method, J_v is needed. Both steps of the partitioning require the results of the other in order to operate correctly. Consequently, a new method has to be found in order to unblock the situation.

5 Resolution methodology

5.1 Robotran working principle

The main components as well as how they interact with each other is shown in Fig.14.

First, the model of the multibody system is implemented on the graphical editor MBSysPad (bodies, joints, anchor points and cuts to close the kinematic loops).

Then, the symbolic files are created by the MBsysTrans module, which are based on the modelled multibody system implemented in MBsysPad.

Next, simulations are conducted in the simulator (called MBsyslab for the Matlab version or MBsysC for the C version), its inputs are the symbolic files generated by MBsysTrans as well as the numerical values for the joints, masses, anchor points, etc. and the type of each joint (dependent, independent or forced/driven).

Finally, after all the wanted simulations have been conducted, the model created in MBsysPad can be animated using the results of the said simulations.

Only the simulator part will be modified in order to implement a solution for mechanisms containing redundant constraints.

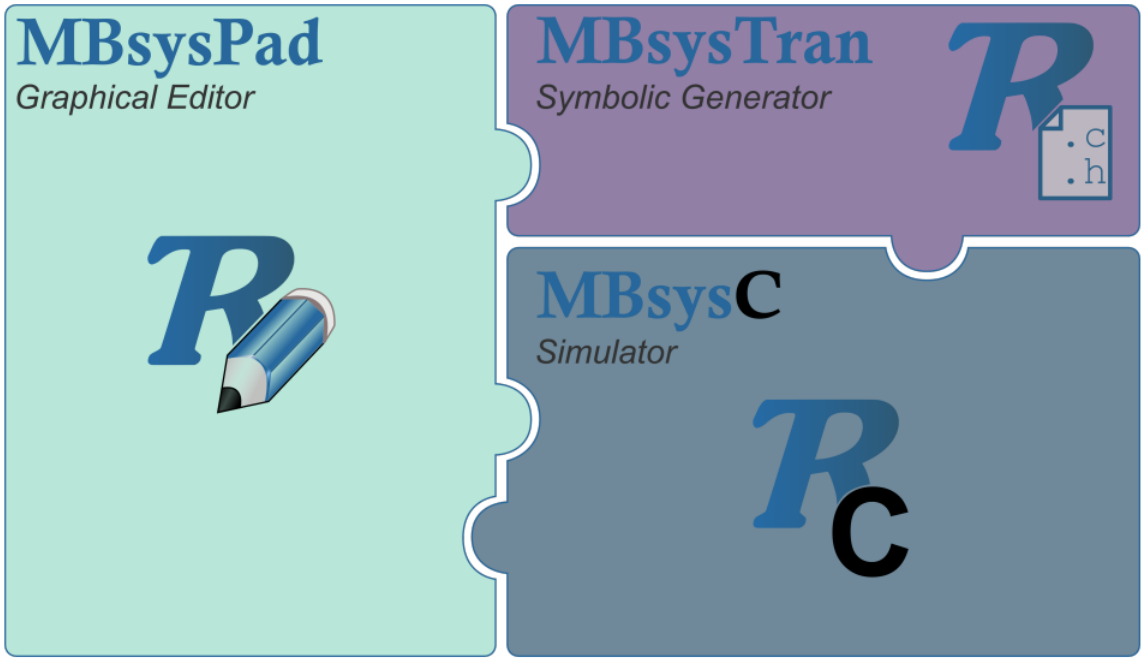


Figure 14: Robotran building blocks. Figure from [4]

5.2 The Non-linear least squares method

The adopted approach consists in finding a way to solve the constraints without having to use the Jacobian J_v. As a reminder, solving the constraints implies that $h(q) = 0$. In other words, what values must have the generalised coordinates q in order for all the components of the vector $h(q)$ to be equal to zero. It can be achieved by using the non-linear least squares method.

This method consists in minimising an objective function F of the form:

$$F = \sum_{1}^{n} r_i(x)^2 \tag{22}$$

where r_i are defined as the residuals and is dependent from the parameters x. It is usually used to find the best fit to non-linear overdetermined systems.
In the case of the constraint solving issue, eq.(22) simply becomes:

$$F = \sum_{1}^{n} h_i(q)^2 \tag{23}$$

Matlab libraries contain some tools in order to solve it, notably the "lsqnonlin" and "fminsearch" functions. Both functions are used in order to minimise the objective

function they are given. The only difference is that fminsearch aims at reducing any form of objective function (not only the form used in the non-linear least squares method) while lsqnonlin is specifically designed to solve the objective function of a non-linear least squares method.
However, before using these tools, a few things must be understood. Firstly, solving the geometrical constraints of a multibody system requires the system to be consistent regarding the constraints (i.e. it is assumed that the system is kinematically sound) and the objective function should end up being very close to zero (or at most of the order 10^{-9}, as it is the precision standard in the case of Robotran).
Furthermore, it is important to have the initial values of q_0 quite close to the values that minimise F, as it is not uncommon with this method to end up in a local minimum instead of the global minimum.
Finally, the only variables in the constraints are the dependent coordinates, as we want to close the loops for fixed values of the independent coordinates.

5.3 Implementation of the method in Robotran

The first step to solve the constraints using lsqnonlin or fminsearch is to define the objective function. It is nothing more than the sum of the square of each individual constraint. Robotran provides these constraints by translating the multibody system "schematic" file- created by using the modelling interface MBSysPad- into symbolic files, one of them containing the symbolic expression of all the constraints. However, it can not be used as it is. In fact, the constraints are written in terms of all the generalised coordinates q, including the forced coordinates and the user's choice for the independent variables. The goal of the constraint solving process is to find the values of the dependent coordinates satisfying the constraints according to fixed values of the user's chosen independent coordinates and forced coordinates. Thus, it is beforehand important to indicate to lsqnonlin and fminsearch which coordinates are fixed and which coordinates are variables, it should then return the values of q_v without changing the values of q_u and q_{driven}.

5.4 Validation of the method

In order to verify the accuracy of this method compared to the Newton-Raphson method, two non-redundant multibody systems were modelled and the simulation was run without including any alteration (the constraints were solved using the endemic Newton-Raphson method).
Then, the values of all the generalised coordinates for all the time steps of the simulation were extracted and saved.
The simulation was then run a second time, but the Newton-Raphson method was

replaced by lsqnonlin and fminsearch successively, the values of all of the generalised coordinates for all the time steps of the simulation in this case were also extracted and saved. Finally, the results computed using Newton-Raphson were compared to the ones computed using fminsearch and lsqnonlin. As a reminder, the error margin is considered small enough if it does not exceed 10^{-9}.

5.4.1 Three bars mechanism

This simple mechanism can be modelled as shown in Fig.15. In this case, it was arbitrarily assumed that $d11 = d31$ and $d01 = d21$ for the sake of being able to easily introduce a redundant constraint in the future. The main frame is located at the starting point of the vector $d01$. This system has three revolute-type generalised coordinates, as well as two prismatic constraints introduced by the ball cut, limiting the system to 1 degree of freedom.

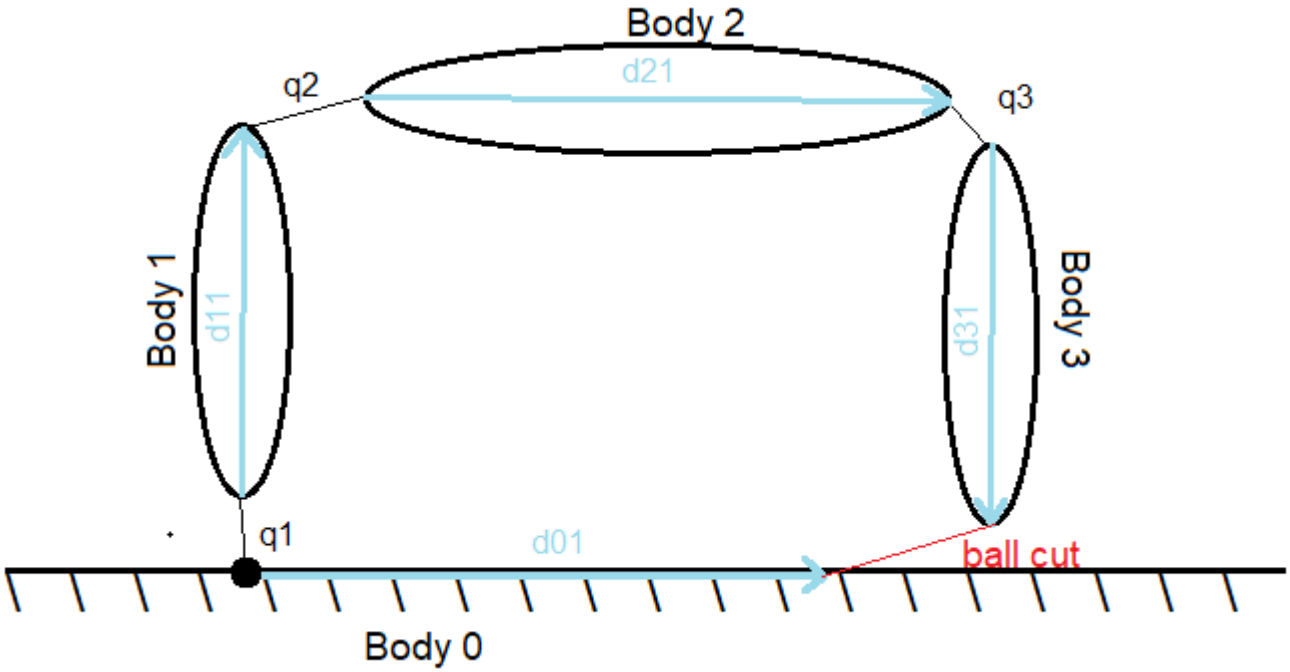


Figure 15: Multibody model of the 3 bars mechanism

After implementing this model into Robotran and having conducted the experiments explained at the beginning of section 5.3, the errors on the 3 generalised coordinates

according to the time step were computed and are shown in Fig.16 and in Fig.17 for fminsearch and lsqnonlin respectively.

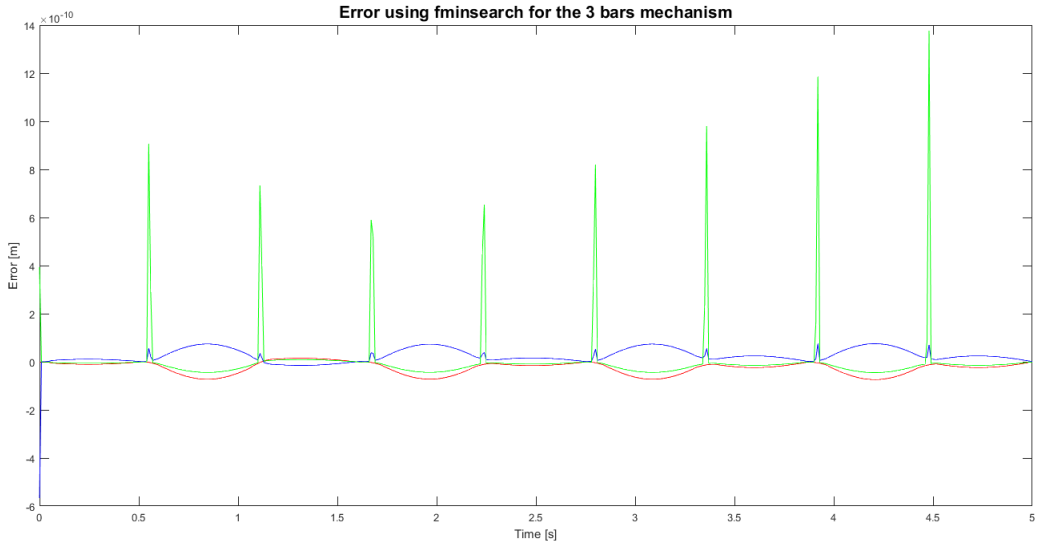


Figure 16: Comparison between Newton-Raphson and fminsearch for the 3 bars mechanism

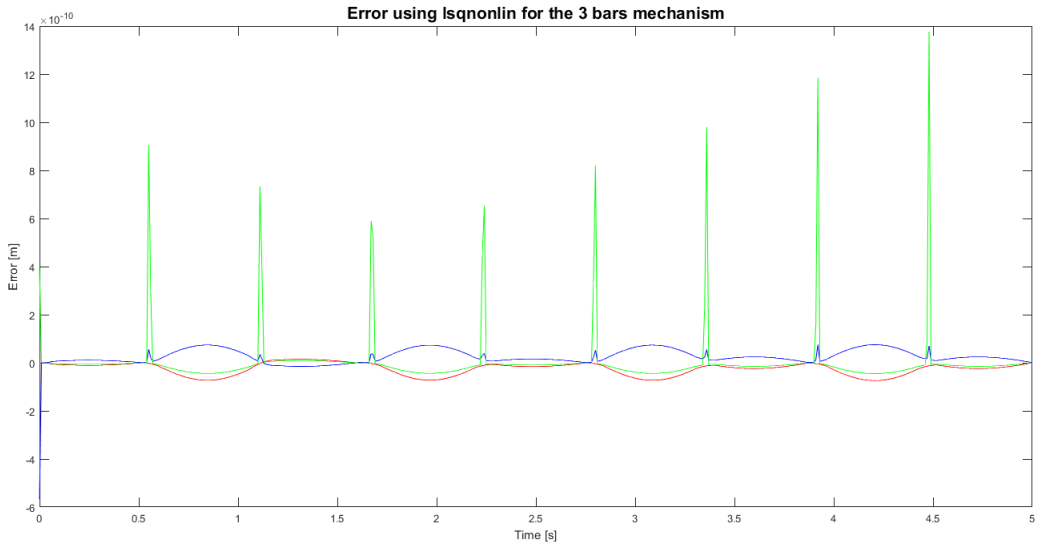


Figure 17: Comparison between Newton-Raphson and lsqnonlin for the 3 bars mechanism

The first thing to notice is that in both cases, the error is lower than the maximum error margin of 10^{-9}. Then, it is also interesting to see that fminsearch and lsqnonlin virtually yield the same results.

5.4.2 Five-point suspension

The last experiment gave quite positive results. However, the analysed system was quite simplistic and two-dimensional. In order to really put this method to the test, a more sophisticated and three-dimensional system is needed. In this case, the mechanism of a Mercedes suspension was chosen. This suspension is a five-point suspension, its schematic is shown in Fig.18; its multibody model is shown in Fig.19 and is taken from [9].

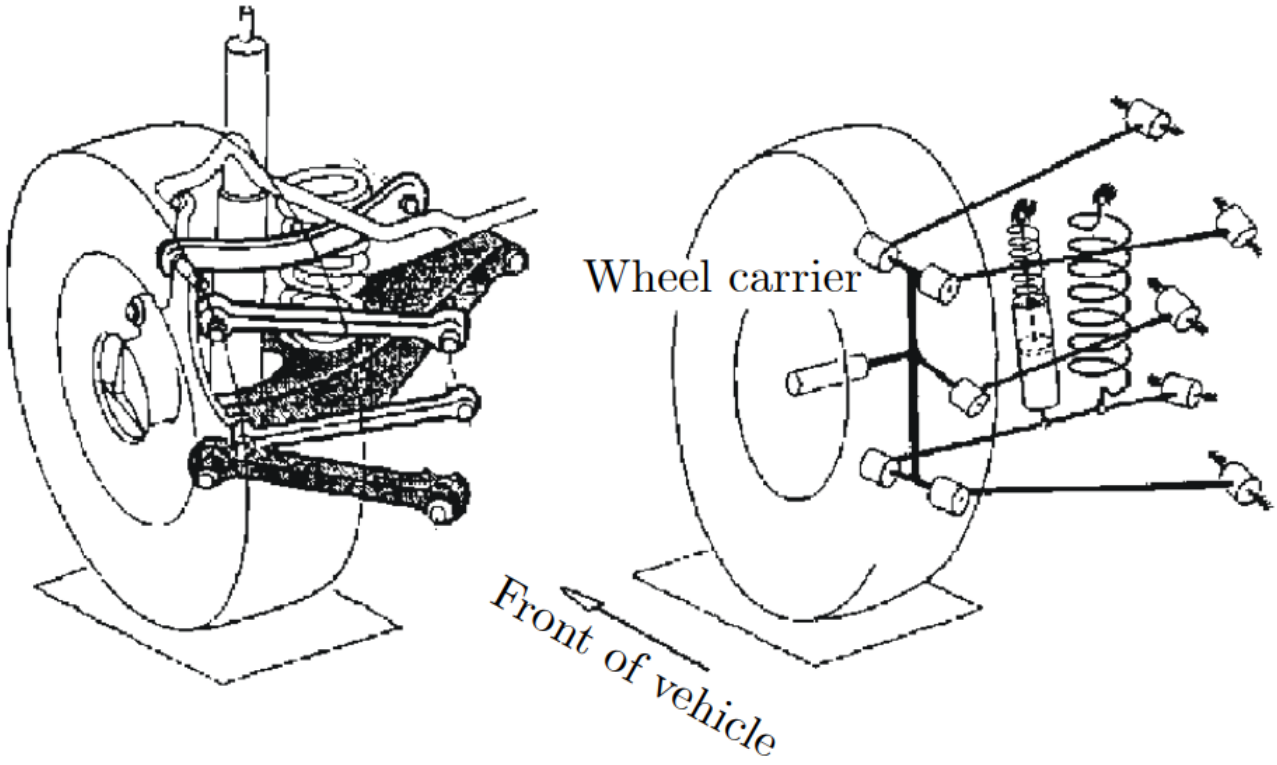


Figure 18: Five-point suspension schematic. Figure from [9]

This mechanism consists in 8 bodies: The chassis, the wheel carrier, the wheel and five bars.
The suspension bars are connected to the chassis using 5 ball joints, the suspension bars are connected to the wheel carrier using 5 ball joints, and the wheel is connected to the wheel carrier using 1 revolute joint. It is also assumed that the bars can not

rotate around their axis, which results in replacing the ball joints connecting the bars to the chassis by 2 d.o.f. universal joints.
With the help of Fig.19, we can see that the system has 2 d.o.f., which is found by subtracting the number of constraints (4 ball-type cut each restraining 3 d.o.f.) from the number of joints (14), these 2 d.o.f. represent the wheel rotation and the displacement of the suspension.

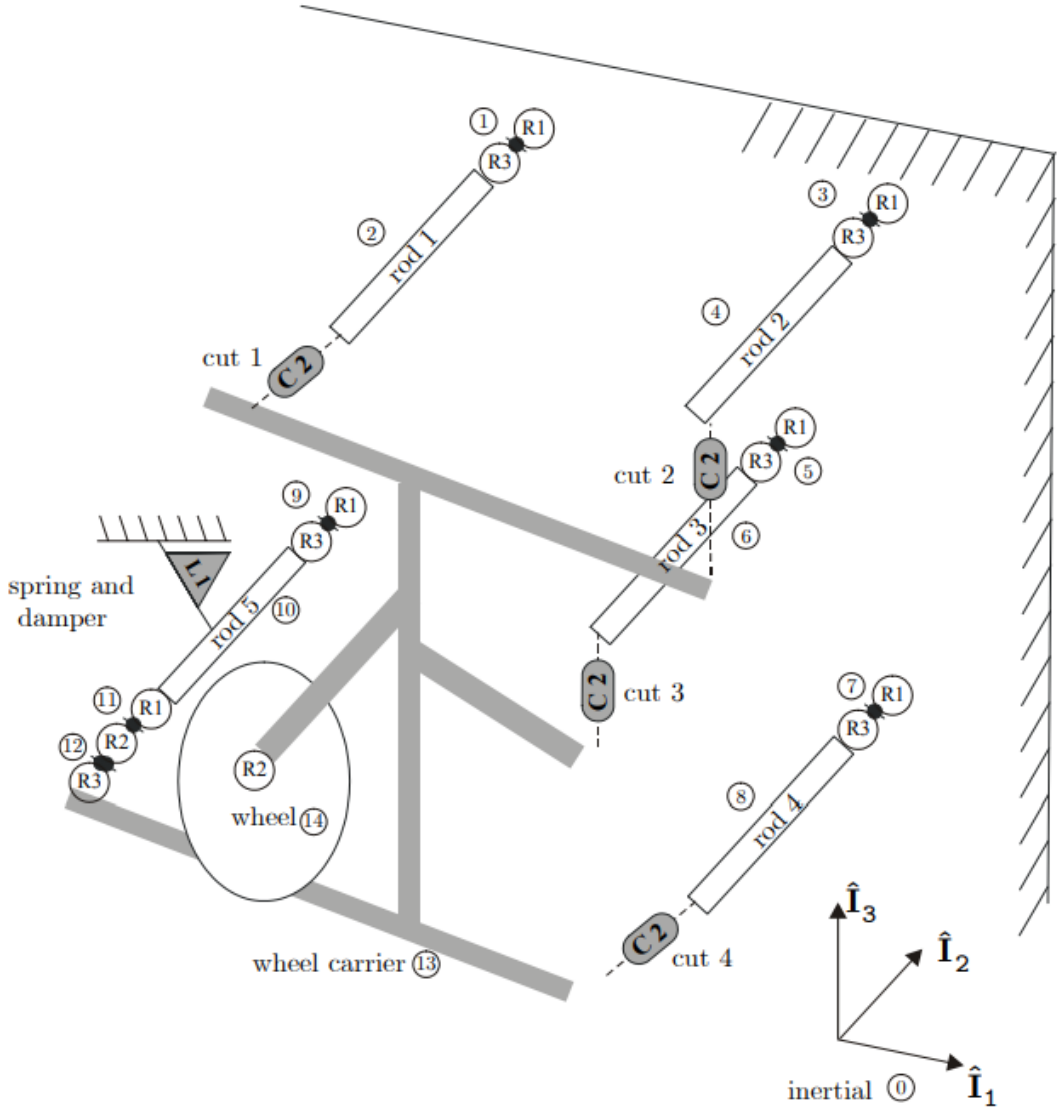


Figure 19: Multibody model of a five-point suspension. Figure from [9]

Once again, the experiments were performed on the model and the error on the

generalised coordinates are shown in Fig.20 and in Fig.21 for fminsearch and lsqnonlin respectively.

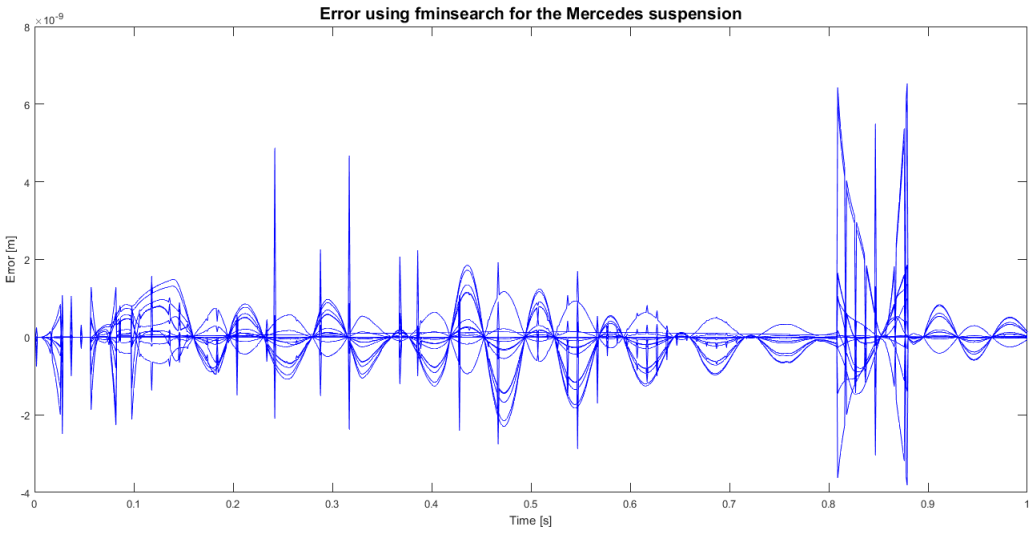


Figure 20: Comparison between Newton-Raphson and fminsearch for the Mercedes

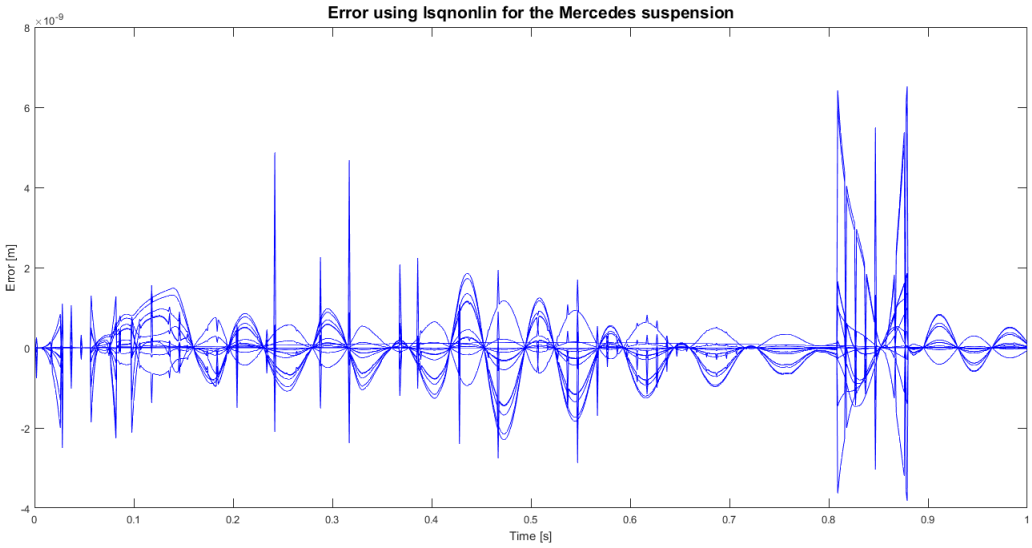


Figure 21: Comparison between Newton-Raphson and lsqnonlin for the Mercedes

The same observations can be made as for the 3 bars mechanism: the error is lower than the tolerance of 10^{-9} and both methods give almost indistinguishable results.

5.4.3 Comment on the results

These observations showed that the non-linear least squares method is accurate enough to solve the constraints of a multibody system. It can thus be used in order to solve the constraints of a multibody system containing redundancies without the need to initially compute J_v.
A side observation that was made during the simulations also showed that the Newton-Raphson method is much faster than the non-linear least squares method. It should however not cause any substantial issue regarding the speed of the process as in practice, when trying to solve a system containing redundancies, the non-linear least squares method will only be used once just before the LU factorisation and the rest of the coordinate partitioning will ensue normally (Newton-Raphson will be used for the rest of the process).

5.5 LU factorisation

Now that the loops are closed, modifications to the LU factorisation part of the Robotran program must be made. Currently, Robotran checks the validity of the factorisation by checking the rank of the obtained Jacobian sub-matrix J_v and comparing it to the rank of the whole Jacobian. Should they be different, J_v will be considered singular and an error occurs. The program implementing the LU factorisation has two possible outcomes:

- The whole process is conducted successfully and returns a rank for the Jacobian that is equal to the maximum between the number of rows (constraints) and columns (generalised coordinates) of the Jacobian.

- While searching for the biggest pivot for the i element of the diagonal, it can only find a zero Pivot. In that case, a redundant constraint has been found and the returned rank is equal to $(i - 1)$.

To understand what happens when a redundant system is introduced to the program, an example is helpful, we will assume the following system:

$$Jac = \begin{array}{c} \\ h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \end{array} \begin{array}{c} \begin{array}{ccccc} q_1 & q_2 & q_3 & q_4 & q_5 \end{array} \\ \begin{pmatrix} \frac{\delta h_1}{dq_1} & \frac{\delta h_1}{dq_2} & \dots & & \\ \frac{\delta h_2}{dq_1} & & & & \\ \vdots & & & & \\ \vdots & & & & \\ & & & & \\ & & & & \end{pmatrix} \end{array} \tag{24}$$

$$Rank(Jac_{unsolved-constraints}) = 5 \tag{25}$$

$$Rank(Jac_{solved-constraints}) = 4 \tag{26}$$

We assume that the system has 6 constraints and 5 generalised coordinates. However, although the system might seem to have too many constraints, it has 1 degree of freedom, which means there are two redundant constraints. If the constraints are not previously solved, it will result in a rank of 5 for the Jacobian and one of these will happen:

- If the user has chosen to define an independent variable by himself, q_5 for instance, the 5^{th} column is locked and the LU factorisation program will end up factorising a 6 by 4 matrix, $Jac_{LU-Fact}$, and its rank will be 4 at most, which is less than 5 and will result in the singular matrix error.

$$Jac_{LU-Fact} = \begin{array}{c} \\ h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \end{array} \begin{array}{c} \begin{array}{cccc} q_1 & q_2 & q_3 & q_4 \end{array} \\ \begin{pmatrix} \frac{\delta h_1}{dq_1} & \frac{\delta h_1}{dq_2} & \dots & \\ \frac{\delta h_2}{dq_1} & & & \\ \vdots & & & \\ \vdots & & & \\ & & & \\ & & & \end{pmatrix} \end{array} \tag{27}$$

$$Rank(Jac_{LU-Fact}) = 4 \tag{28}$$

$$Rank(Jac_{LU-Fact}) < Rank(Jac_{unsolved-constraints}) \tag{29}$$

- If the user decides to let the program choose the independent variable, the LU factorisation will be successful and the diagonal will have the largest pivot, but as no zero pivot were found during the factorisation, the Jacobian of the independent variable will be empty, which will also result in an error.

$$Jac_{LU-Fact} = Jac \tag{30}$$
$$Dim(J_v) = [5, 5] \tag{31}$$
$$Dim(J_u) = [] \tag{32}$$
$$\tag{33}$$

Now, it is assumed that the constraints have previously been solved, the Jacobian will thus have a rank of 4, as 2 lines in the [6∗5] Jacobian will be linear combinations of the others. Now, one of these will happen:

- If the user has chosen to define an independent variable by himself, the LU factorisation program will end up factorising a 6 by 4 matrix and its rank will be 4, which is equal to the computed rank of the complete Jacobian, the returned sub-matrix J_v will thus be an acceptable and regular solution.

$$Rank(Jac_{solved-constraints}) = 4 = Rank(Jac_{LU-Fact}) \tag{34}$$

- If the user decides to let the program choose the independent variable, the LU factorisation will stop at the 5^{th} element of the Jacobian diagonal because it stumbled upon a zero pivot. It will show that there are redundant constraints, will proceed to ignore the remaining rows and will return J_v, a 4 by 4 sub-matrix of rank 4 of the current state of the factorised matrix,$Jac_{Factorised}$, it will thus also be seen as an acceptable and regular solution.

$$Jac_{Factorised} = \begin{array}{c} \\ h_6 \\ h_3 \\ h_4 \\ h_1 \\ h_5 \\ h_2 \end{array} \begin{array}{c} \begin{array}{ccccc} q_5 & q_1 & q_2 & q_3 & q_4 \end{array} \\ \begin{pmatrix} . & ... & ... & & \\ \vdots & & & & \\ \vdots & & & & \\ \vdots & & & & \\ \vdots & & & & \\ & & & & 0 \end{pmatrix} \end{array} \tag{35}$$

$$J_v = Jac_{Factorised}(1:4, 1:4) \tag{36}$$
$$J_u = Jac_{Factorised}(1:4, 5) \tag{37}$$

The whole process having been made clearer, the only modification to make in the program, regarding the LU Factorisation, is to allow it to return the current sub-matrix J_v when the factorisation stumbles upon a zero pivot. Furthermore, it is also important to allow the row permutations, as it is not activated by default.

5.6 Summary

When launching the simulation after having implemented the redundant constraints, the following will happen:

- **Before the simulation:** It is assumed that the symbolic files have previously been generated using the MBSysPad model.
- **Step 1:** The program will load the data of the system from the MBSysPad model (dimension, mass, forced joints, user's choice for independent coordinates, etc.).
- **Step 2:** The program will begin the coordinate partitioning.
- **Step 3:** The program will induce randomness to coordinates set to zero in order to avoid any singularities.
- **Step 4:** Either fminsearch or lsqnonlin will be used in order to solve the constraints, it is important to verify that the given function to minimise only depends on the dependent variables (the independent and forced variables are fixed).
- **Step 5:** The program computes the rank of the Jacobian with solved constraints. If there are any redundant constraints, this rank will be lower than if the constraints were not solved.
- **Step 6:** The program starts the LU Factorisation.
- **Step 6.a:** If during the factorisation, a zero pivot is met, the factorisation stops and returns the current factorised square sub-matrix J_v in such a way that the line containing the zero as well as the following lines are ignored. It also returns its rank and the permutations that have been done
- **Step 6.b:** If no zero pivot has been found, it returns the completely factorised sub-matrix J_v as well as its rank.
- **Step 7:** If the sub-matrix J_u is empty or if the rank of J_v is lower than the rank of the whole Jacobian (also possible if the constraints have not been solved, e.g. bad user choice for independent coordinates), returns an error. If not, the program continues to step 8

- **Step 8:** The program adds the potential new independent coordinates and a second partitioning is performed in order to check the validity of the choice.
- **Step 9:** The coordinate partitioning ends and the program can start the temporal simulation.

Here, step 4 is totally new and step 6.b requires the code to be somewhat modified. The modifications brought to the Matlab code are detailed in Annex 1.

6 Application

Now that a potential solution has been found and implemented, it is time to put it to the test. In the following section, two applications will be presented. These applications aim to check if the performed changes allow Robotran to deal with redundant constraints. The first application consists in a simple two-dimensional mechanism, while the second one consists in a complex three-dimensional mechanism.

6.1 Redundant 4 bars mechanisms

6.1.1 Description of the mechanism

The mechanism that is dealt with in this application is the one shown in Fig.12. Its model (see Fig.22) is a variation of the model shown in Fig.15, the difference being that there is an additional bar. In this case, we have $d02 = d22$ and $d11 = d31 = d41$. Furthermore, it is assumed that $d02$ and $d01$ are aligned, the same goes for $d22$ and $d21$. The 4 joints are single d.o.f. revolute joints and the system is cut using joint cuts of type ball (2 constraints in 2D) between Body 4/Body 3 and Body 0.

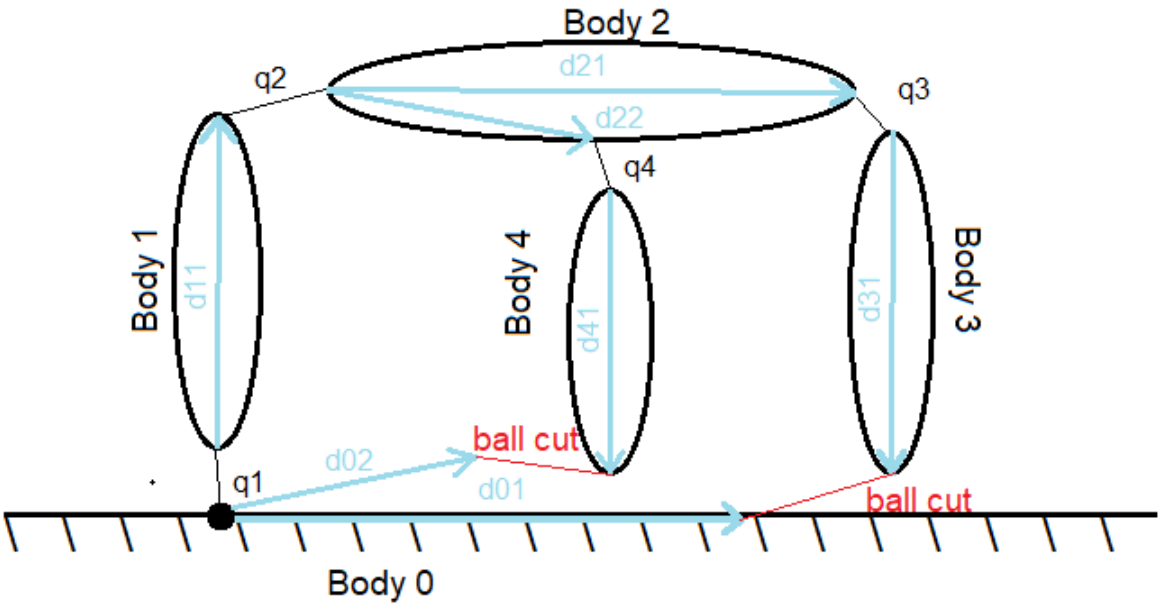


Figure 22: Multibody model of the 4 bars mechanism

As already explained for Fig.12, computing the number of d.o.f. of this system using classical methods would result in 0 d.o.f., however the correct number of d.o.f.

is in fact one, There is a redundant constraint and this system can not be solved using the regular version of Robotran (if the system is not already closed).

6.1.2 System data

There is an infinite number of possibilities to set the value for the masses, joints and anchor points. For this application, the following input were chosen:

Joints:

	Initial value [rad]	Type
q1	0.78539816339	u
q2	1.0	v
q3	0.1	v
q4	0.4	v

Anchor points:

	Component	Value [m]
d01	X	1.0
d02	X	0.5
d11	Z	0.5
d21	Z	1.0
d22	Z	0.5
d31	X	0.5
d41	X	0.5

Masses: As we are more interested in the kinematics than in the dynamics of the system, we assumed that every bar weighs 1[kg], the inertia matrices are assumed to be zero matrices and the centers of mass are located in the middle of each bars.

6.1.3 Results

We let the 4 bars oscillate like a pendulum (we assumed that body 0 was the ceiling, we just have to imagine that Fig.22 is upside down). The program was able to detect and eliminate the redundant constraint, the position of the joints through time are shown in Fig.23. Both fminsearch and lsqnonlin ended up giving the same result.

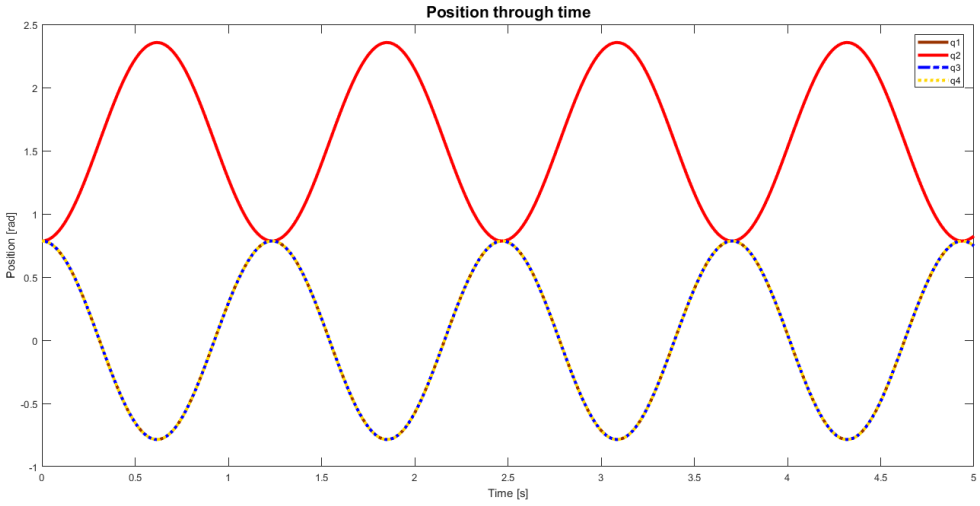


Figure 23: Obtained result for the 4 bars

This application allowed us to confirm that the implemented method is valid, or at least, for simple 2-D problems.

6.2 Agile eye

6.2.1 Description of the mechanism

As mentioned in section 4.1, a more complex mechanism containing redundant constraints would be the agile eye (see Fig.13). The agile eye was introduced in 1994 by Clément Gosselin [7]. It is a parallel robot consisting of 3 motors, 6 arms and the "eye". Its purpose is the fast and accurate orientation of cameras, although it can also be used to orientate other objects, such as lasers, mirrors, etc. It is able to achieve angular velocities of more than 1,000 degrees per second and accelerations of more than 20'000 degrees per second square with minimal error [8].

Its architecture is defined in such a way that all 9 joints of the system have their main axis intersect at the same point: the center of rotation of the eye [8]. Attempts were made to implement this system in Robotran, this could however not be achieved due to the redundant constraints not being implemented in the program.

The mechanism is symmetrical; there is the eye as well as three identical set of bodies. One set contains: a motor whose rotation axis points toward the center of rotation, the first arm connected to the motor, the second arm connected to the first arm by a joint whose axis coincides with the axis of the neighbouring motor,

this second arm is finally connected to the eye in such a way that the axis of the joint coincides with the axis of the other neighbouring motor. The geometry of these arms must allow them not to interfere with any other body of the system. However, as the model implemented in Robotran is not "solid", we can choose the geometry of these arms in a way that suits us (the bodies will pass through each other).

The multibody model of the agile eye as implemented in Robotran is shown in Fig.24.

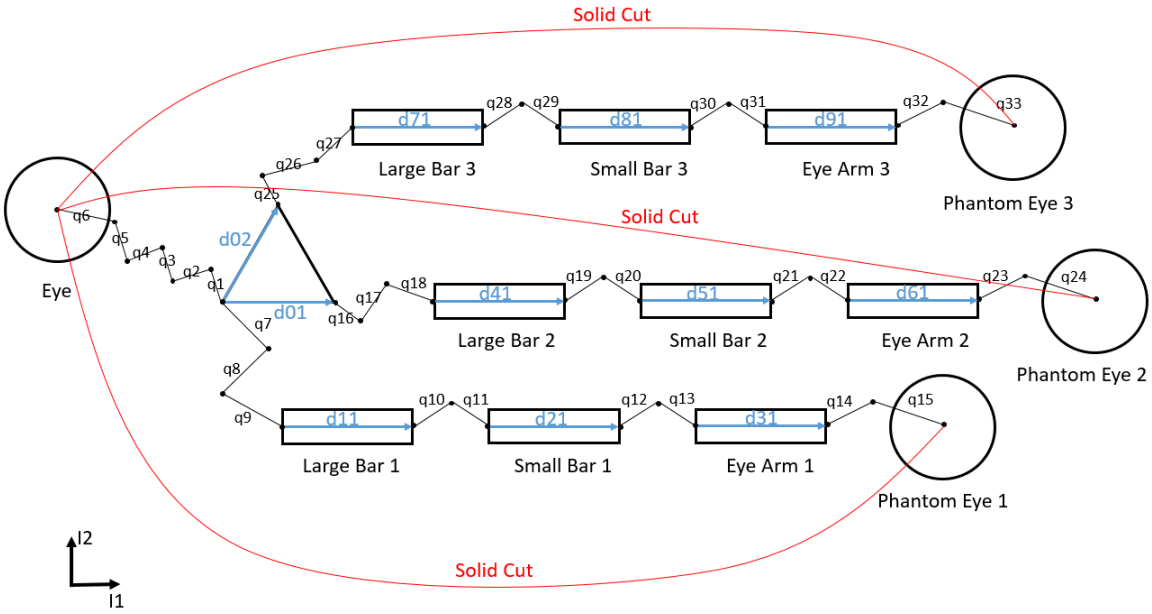


Figure 24: Model of the agile eye as implemented in Robotran

The model consists of the eye and 3 identical branches. These branches end with an eye arm, which represents the part of the eye connected to the last bar of each branch, and a phantom eye, which is a phantom body used in order close the loops using solid cuts (constraint that aims to lock the 6 degrees of liberty on a certain frame,i.e. the eye).

6.2.2 System data

An agile eye can have many different architectures as long as all of its 9 joints have their axis intersect the center of rotation. We chose to characterise the architecture of our agile eye with the help of 2 parameters: the length L of the sides of the equilateral triangle formed by the motors, and the angle θ between the axis of these motors and the ground. For this particular simulation, $L = 1$ [m] and $\theta = \frac{\pi}{4}$ [rad] were chosen.

Joints:

	Symbolic value	Initial value [rad] or [m]	Motion	Type
q1	[/]	0.4	T1	v
q2	[/]	0.28	T2	v
q3	[/]	0.55	T3	v
q4	[/]	0.0	R1	v
q5	[/]	0.0	R2	v
q6	[/]	0.0	R3	v
q7	$\frac{\pi}{6}$	0.5235987755982989	R3	Forced
q8	$-\theta$	-0.7853981633974483	R2	Forced
q9	[/]	-0.68471920311242	R1	u
q10	$-\beta$	-1.318116071652818	R2	Forced
q11	[/]	0.2	R1	v
q12	$\frac{\pi}{2} - \beta$	0.25268025514207865	R3	Forced
q13	[/]	2.25	R2	v
q14	θ	0.7853981633974483	R1	Forced
q15	0	0.0	R3	Forced
q16	$\frac{5*\pi}{6}$	2.6179938779914944	R3	Forced
q17	$-\theta$	-0.7853981633974483	R2	Forced
q18	[/]	-0.68471920311242	R1	u
q19	$-\beta$	-1.318116071652818	R2	Forced
q20	[/]	0.2	R1	v
q21	$\frac{\pi}{2} - \beta$	0.25268025514207865	R3	Forced
q22	[/]	2.25	R2	v
q23	θ	0.7853981633974483	R1	Forced
q24	$-\frac{2\pi}{3}$	-2.0943951023931957	R3	Forced
q25	$-\frac{\pi}{2}$	-1.5707963267948966	R3	Forced
q26	$-\theta$	-0.7853981633974483	R2	Forced
q27	[/]	-0.68471920311242	R1	u
q28	$-\beta$	-1.318116071652818	R2	Forced
q29	[/]	0.2	R1	v
q30	$\frac{\pi}{2} - \beta$	0.25268025514207865	R3	Forced
q31	[/]	2.25	R2	v
q32	θ	0.7853981633974483	R1	Forced
q33	$\frac{2\pi}{3}$	2.0943951023931957	R3	Forced

It is important to remember that these values are written according to the con-

cerned body frame, not systematically the base frame. It is also to be noted that the initial value of a forced joint is the value that this joint will have for the entirety of the simulation. A symbolic value was also given to the forced joint according to the parameters L and θ. Furthermore, to somewhat condense the expression, we have $\beta = 2 * asin(\frac{3*cos(\theta)}{2\sqrt{(3)}})$

Anchor points:

	Component	Symbolic value	Input value [m]
d01	X	L	1.0
	Y	0	0.0
	Z	0	0.0
d02	X	$\frac{L}{2}$	0.5
	Y	$\sqrt{3} * \frac{L}{2}$	0.866025403784439
	Z	0	0.0
d11	X	$L * cos(\frac{\pi-\beta}{2})$	0.6123724356957945
	Y	0	0.0
	Z	$-L * sin(\frac{\pi-\beta}{2})$	-0.7905694150420949
d21	X	$\frac{(2+cos(\beta))*L\sqrt{3}}{6*cos(\theta)}$	0.9185586535436918
	Y	$-\frac{sin(\beta)*L\sqrt{3}}{6*cos(\theta)}$	-0.39528470752104744
	Z	0	0.0
d31	X	0	0.0
	Y	$\frac{L\sqrt{3}}{6*cos(\theta)}$	0.408248290463863
	Z	0	0.0

For the 6 remaining anchor points, we just have to remember that $d11 = d41 = d71$, $d21 = d51 = d81$ and $d31 = d61 = d91$.

Masses:

Once again, as we are more focused on the kinematics rather than in the dynamics of the system, we assumed every bodies of the system to have masses and inertia equal to zero. The only body with a mass and inertia will be the body "Eye", it will be considered as a full sphere of radius $r = 0.2$[m] and with a mass $m = 4$[kg], the element on the diagonal of the matrix of inertia will thus be $I = \frac{2}{5}mr^2 = 0.064[kg * m^2]$.

6.2.3 Results

This time, only lsqnonlin was able to solve the constraints. It seems logical, as the system to solve was quite complex and lsqnonlin is specifically designed to handle this kind of problem, unlike fminsearch. The program was able to solve the constraints both with user's choice of independent variable and without.

In the first simulation, the same sinusoidal torque was given to the 3 motors, the result are shown in Fig.25. As the 3 motors are given the same torque, the eye must rotate around its vertical axis, which is the case.

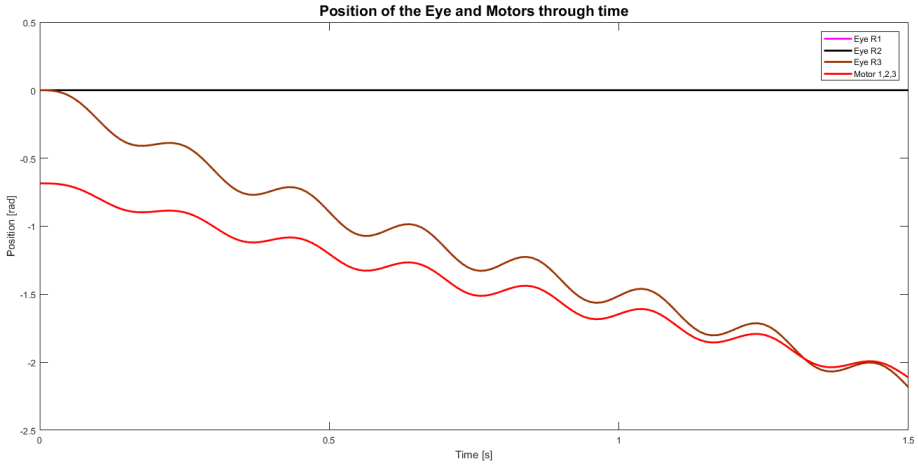


Figure 25: Result of simulation 1

In the second simulation, 2 of the motors were given the same sinusoidal torque, while the remaining one was given a different sinusoidal torque. In this case, the eye rotates around its 3 axis, the results are shown in Fig.26

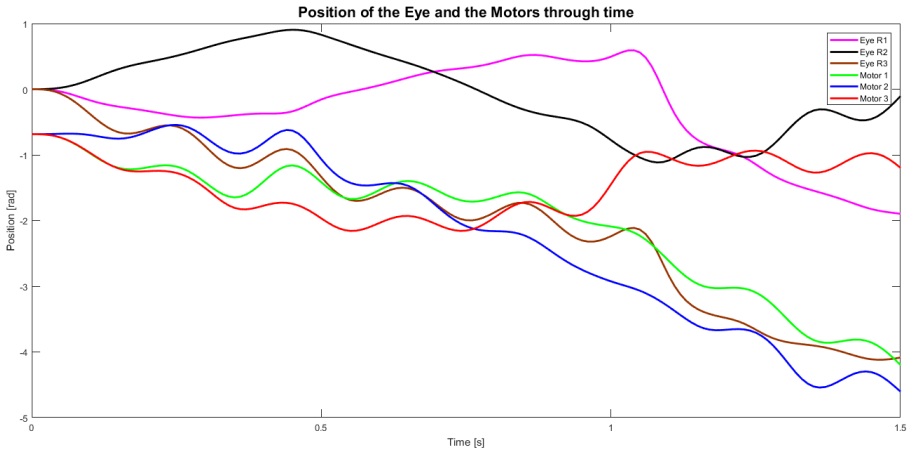


Figure 26: Result of simulation 2

These results show us that the modelling of the agile eye is now possible and a wide range of simulations, e.g. the control of the motors, is now available.

6.3 Comments on the results

The implementation of the redundant constraints has been a success, the program should work for other systems containing redundant constraints. It is however important to point out that after a certain degree of complexity of the system, fminsearch becomes quite unstable and lsqnonlin becomes the preferred alternative.

7 Conclusion

To summarise what has been done in this thesis, the non linear least squares method has been implemented as a complement to the already existing coordinate partitioning method in order to detect and handle redundant constraints in multibody systems. The goal that has been achieved is to add the possibility to detect those constraints even if the multibody system has not initially been closed correctly.

If we want to look into what can practically be done from now on with the implemented method, the Robotran user will now be able to model mechanisms containing redundant constraints without any difficulties, a concrete example being the agile eye. Furthermore, should someone try to deal with redundant constraints with anything other than Robotran, he will know that the non-linear least squares method is a viable solution to solve constraints.

Regarding further topics that are worth looking into now that this issue has been tackled, the implementation of a program able to compute the constraint forces in the eliminated redundant constraint, as Marek Wojtyra did [2], would be particularly interesting. Furthermore, there may be ways to implement the redundant constraints other than optimisation methods.
Even more concrete issues can be looked into: Robotran also has a C version and a Python version. Implementing a solution in those versions would make it accessible for a much wider range of people.
As modelling the agile eye is now possible, a lot of different analysis can be made, e.g. the control of the motors, the effect of some parameters on its performance, etc.
As we can see, a lot can still be done. However, the results obtained in this thesis could be of some value as a means to achieve these goals.

Bibliography

[1] Maria Augusta Neto and Jorge Ambrósio. Stabilization methods for the integration of dae in the presence of redundant constraints. *Multibody System Dynamics*, 10(1):81–105, 2003.

[2] Marek Wojtyra. Joint reaction forces in multibody systems with redundant constraints. *Multibody System Dynamics*, 14(1):23–46, 2005.

[3] J García de Jalón and Maria Dolores Gutierrez-Lopez. Multibody dynamics with redundant constraints and singular mass matrix: existence, uniqueness, and determination of solutions for accelerations and constraint forces. *Multibody System Dynamics*, 30(3):311–341, 2013.

[4] Quentin Docquier Paul Fisette. Modeling multibody systems with robotran. `https://www.robotran.be/images/files/Robotran_basics.pdf`. Accessed: 2020-06-18.

[5] Paul Fisette. Epl course; lmeca2802:"multibody system dynamics". Accessed: 2020-06-18.

[6] Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU press, 2012.

[7] C. M. Gosselin and J. . Hamel. The agile eye: a high-performance three-degree-of-freedom camera-orienting device. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 781–786 vol.1, 1994.

[8] C. M. Gosselin, E. St. Pierre, and M. Gagne. On the development of the agile eye. *IEEE Robotics Automation Magazine*, 3(4):29–37, 1996.

[9] Jean-Claude Samin and Paul Fisette. *Symbolic modeling of multibody systems*, volume 112. Springer Science & Business Media, 2013.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**