Group 27 Project presentation: WINX

Boffo Marco,
Brusadin Giulia,
Camporese Maria,
Morettini Simone

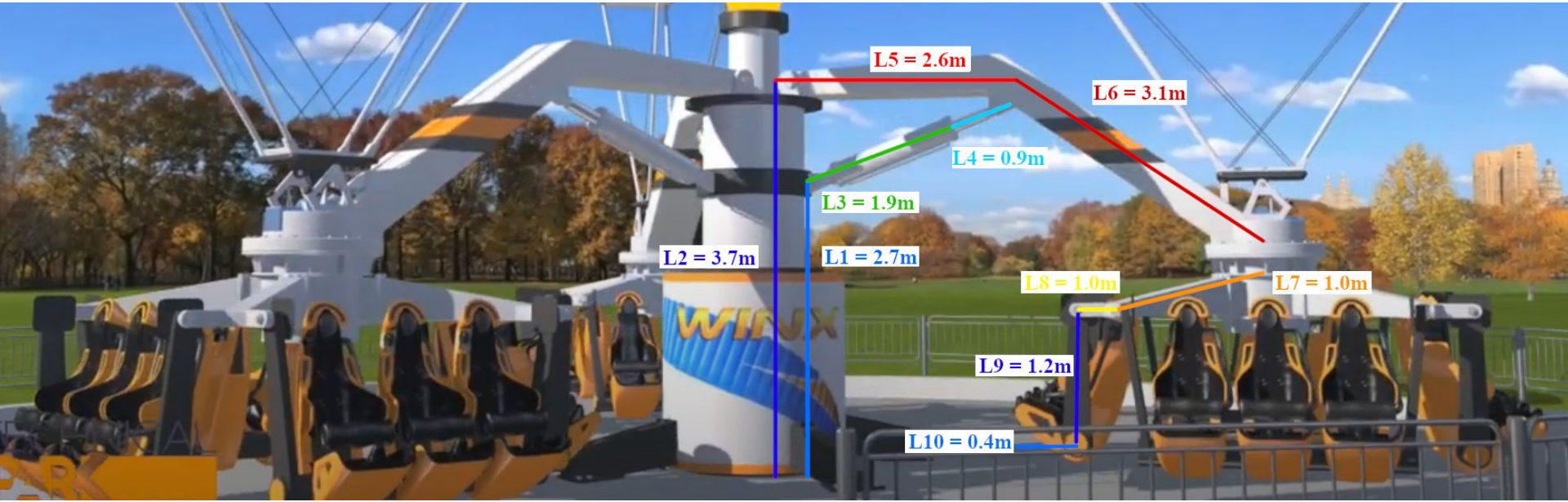
Course of Modelling and Simulation of Mechatronic Systems A.Y. 2019/2020
University of Trento

Index

- **Engineering specifications**
- **Kinematic analysis**
  - Recursive approach
  - Global approach
- **Dynamic analysis**
  - Maple
  - MatLab
  - MapleSim

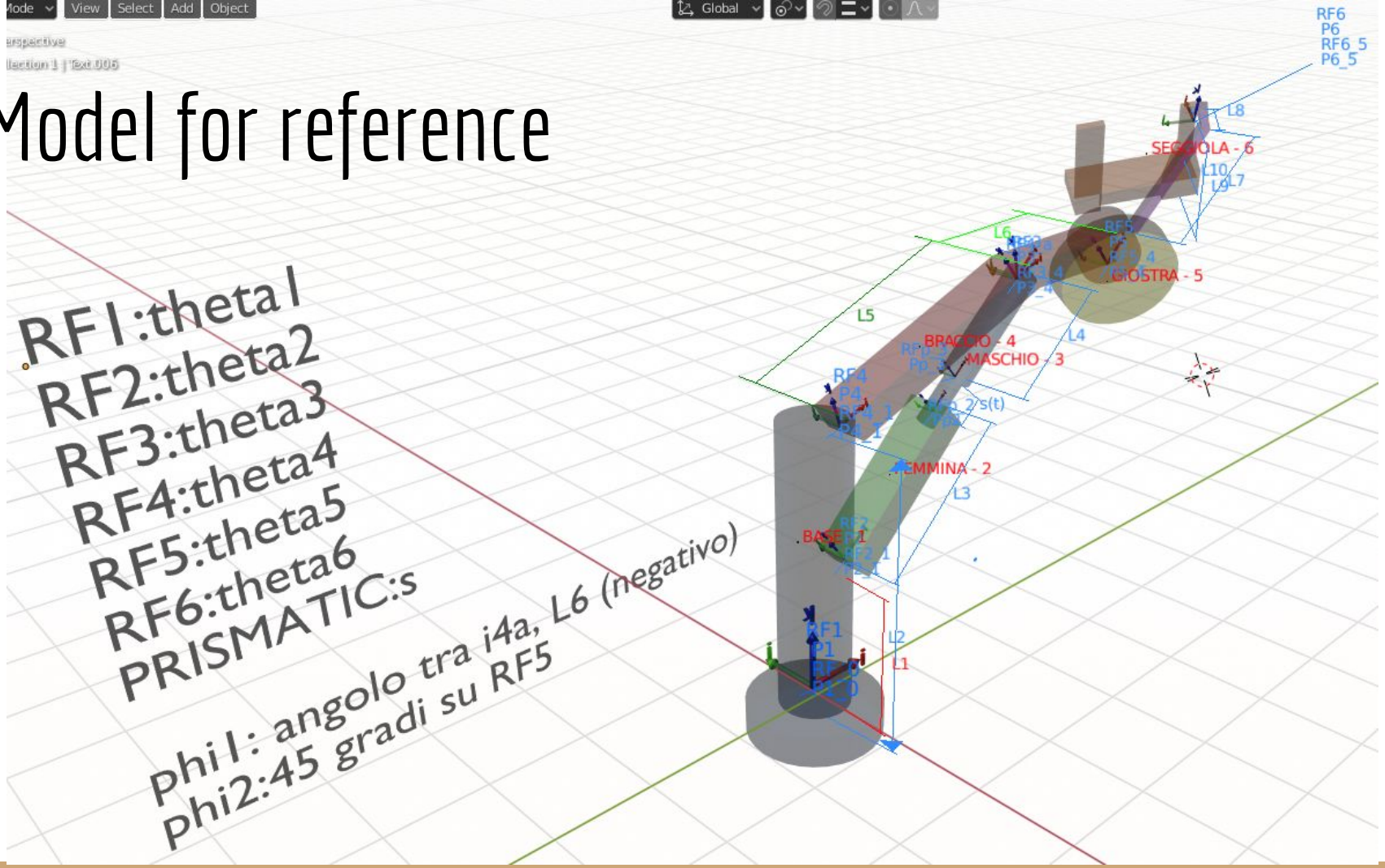
Engineering specifications

Dimensions



We have tried to keep the same proportions of the given 3D rendering and to obtain similar dimension to the ones of a similar attraction

Model for reference



Objectives

System Requirement	Engineer specification	Target performance	Weight
Safety: avoid nausea, vision disruption, cardiovascular problems and give chance to brace	Follow the limits imposed on [Table1]	Acceleration below the values expressed in [Table1]	4
Fun: high speed, atypical physical experience and change the user perspective	Maximize fun function		3
Durability	Diminish stress on the structure	Minimize jerk	2
Occupied space	Contained working space dimension	20x20m (minimize if possible)	1

Ranges of admissible accelerations

Table 1. Admissible amusement rides accelerations (i.e. values for **a–g**) for the x-, y- and z-directions depending on the time of exposure [6].

Admissible acceleration	0.2 s		1.5 s		>12 s	
	min	max	min	max	min	max
a_x	$-2.0g$	$+6.0g$	$-1.5g$	$+5.0g$	$-1.5g$	$+2.5g$
a_y	$-3.0g$	$+3.0g$	$-2.5g$	$+2.5g$	$-2.0g$	$+2.0g$
a_z	$-2.0g$	$+6.0g$	$-1.5g$	$+5.0g$	$-1.1g$	$+2.0g$

[Table1]Eager, David, Ann-Marie Pendrill, and Nina Reistad.
"Beyond velocity and acceleration: jerk, snap and higher derivatives."
European Journal of Physics 37.6 (2016): 065008

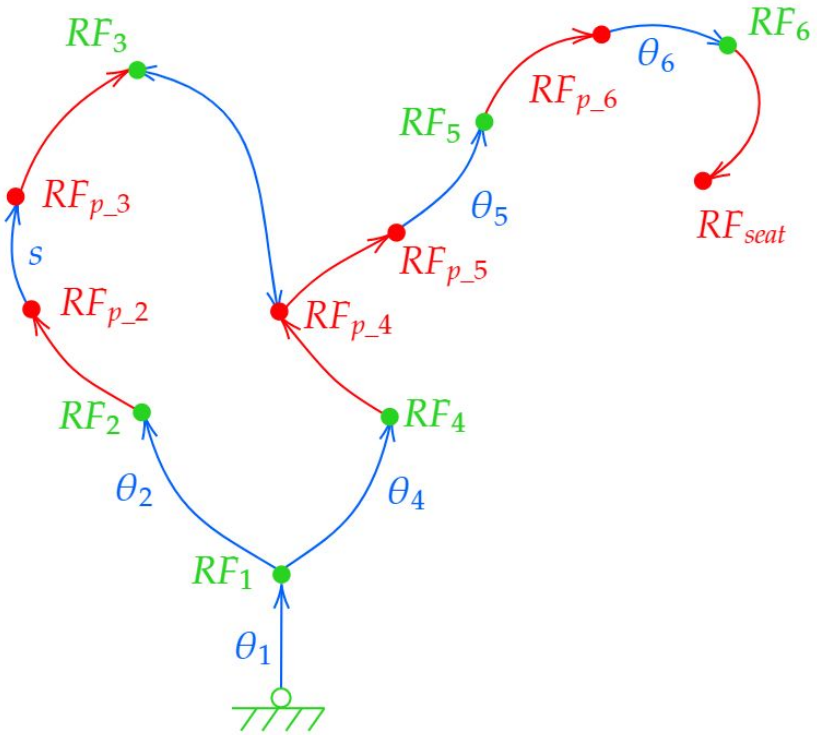
Kinematic analysis

Recursive Approach

We defined the body reference frames in recursive formulation.

Since there is a loop in the system, we added two constraint equations, derived from the overlapping of the origin of two reference frames.

We then used rehonomous constraints in the kinematic analysis, leaving time as the only independent variable.

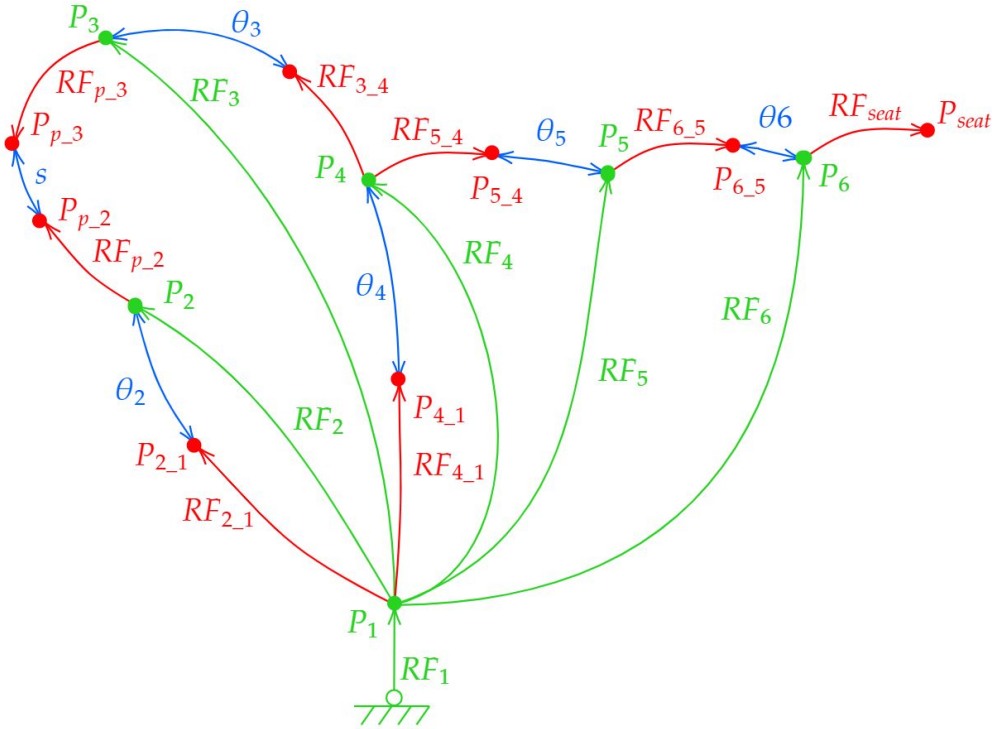


Global Approach

We defined the simplified method, maintaining the proper global formulation.

We obtain, in the end, 21 variables, of which 17 are dependent. Therefore, we have 17 constraint equations.

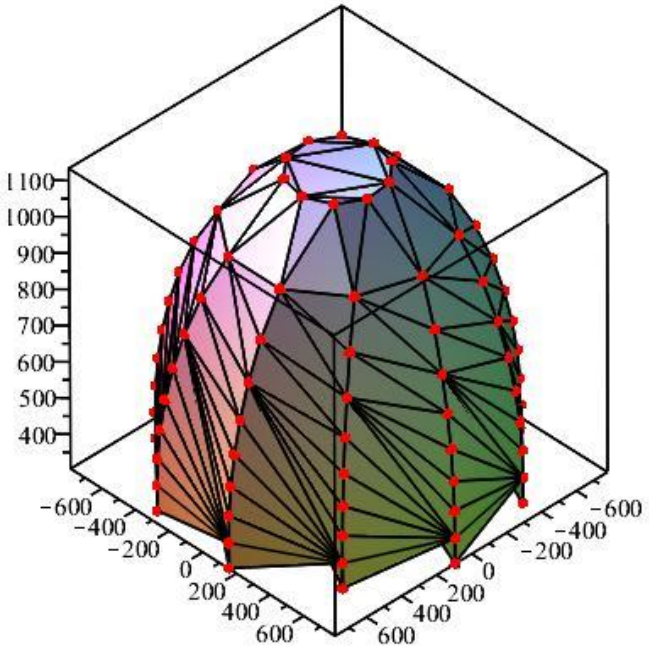
We then used rehonomous constraints in the kinematic analysis, leaving time as the only independent variable.



Workspace

The workspace, calculated as the maximum space occupied by the traction, is:

- along X from -778 cm to 778 cm
- along Y from -787 cm to 787 cm
- along Z from ground to 1133



Initial position problem

Recursive

The system has proven to be simple enough to use the analytical approach.

Global

Due to the complexity and number of the constraints we decided to use the numerical solution.

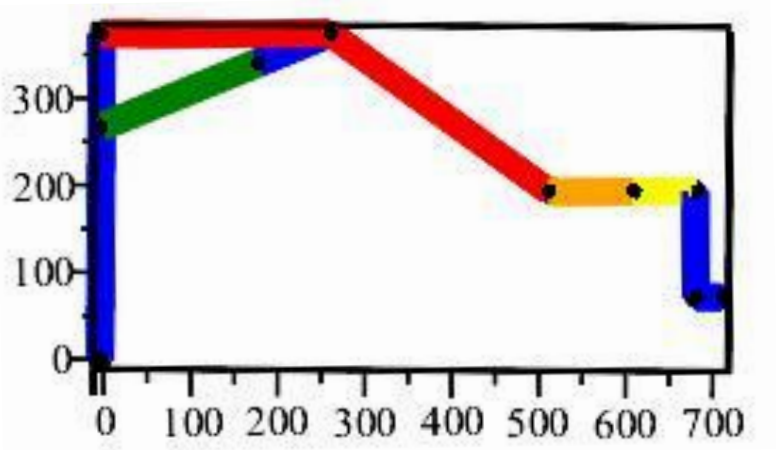
Seat Position at beginning: [m]

$$\begin{bmatrix} 7.102056996 \\ 0.9712111641 \\ 0.7630123080 \end{bmatrix}$$

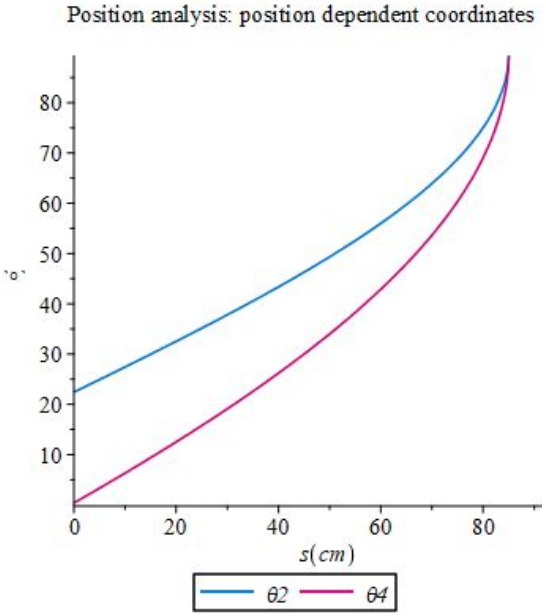
Results

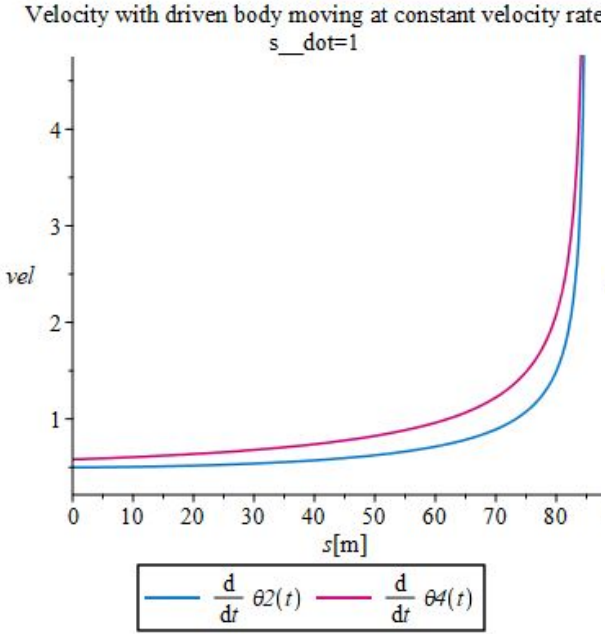
s(t) = 0 cm, θ1(t) = 0 rad, θ5(t) = 0 rad, θ6(t) = 0 rad

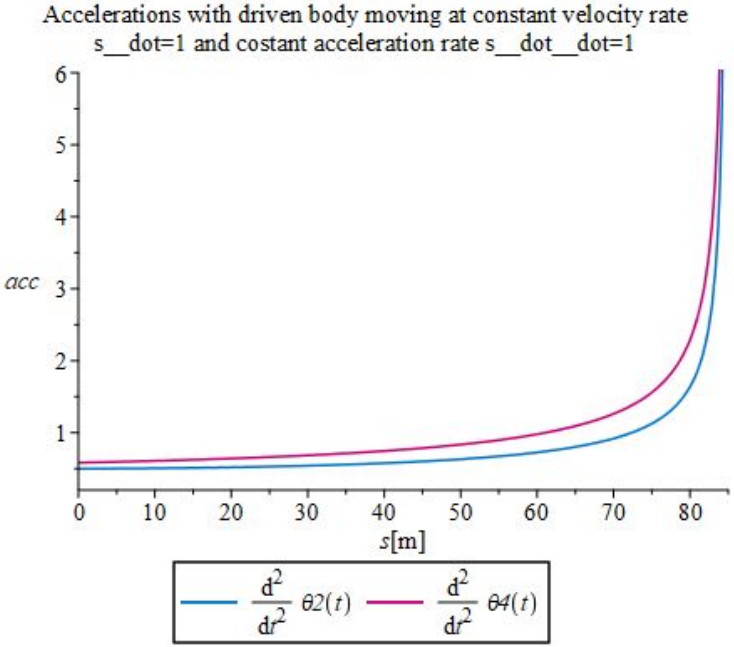
θ2(t) = arcsin(0.381) = 0.391 rad
θ4(t) =arctan(0.00678, 0.999)}= 0.00678 rad



Singular Configuration



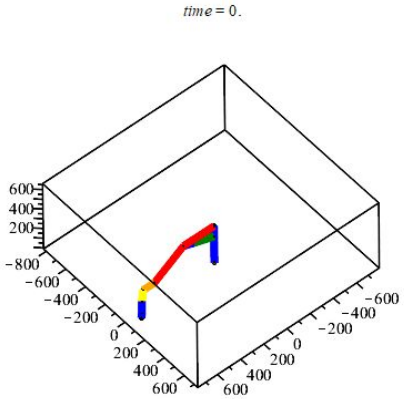




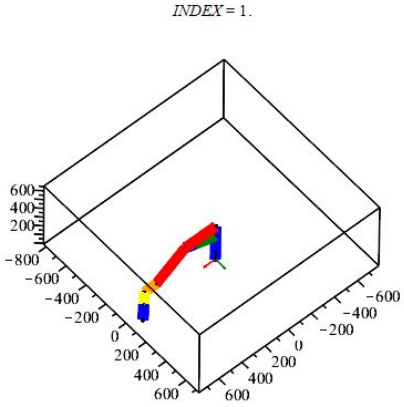
$$\left[\{ s(t) = 85.05000000,\ \theta 1(t) = \theta 1(t),\ \theta 2(t) = 1.570796327,\ \theta 4(t) = 1.570796327,\ \theta 5(t) = \theta 5(t),\ \theta 6(t) = \theta 6(t) \} \right]$$

Profiles Animation

```
s_profile_time := alpha__s - alpha__s*cos(omega__s*t);
theta1_profile_time := omega__1*t;
theta5_profile_time := omega__5*t;
theta6_profile_time := alpha__6*sin(omega__6*t);
```



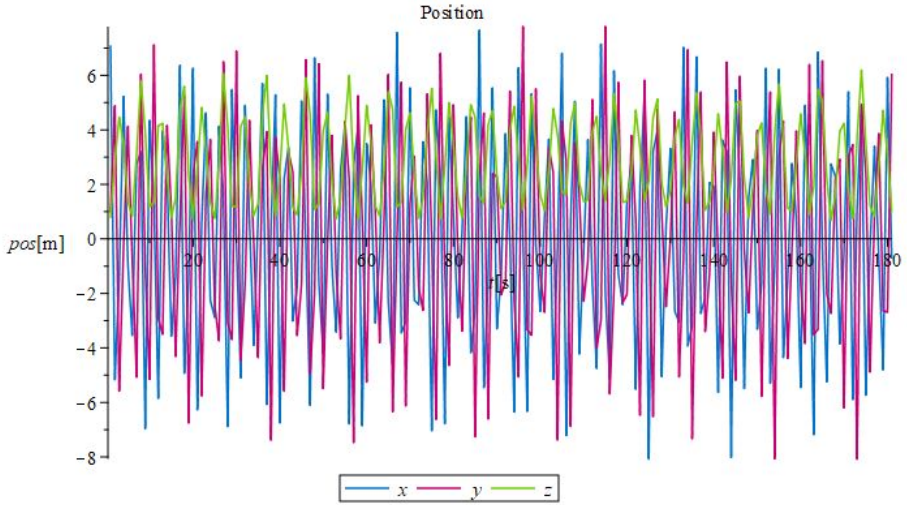
Recursive approach
(Analytical of 30 of 180 sec)



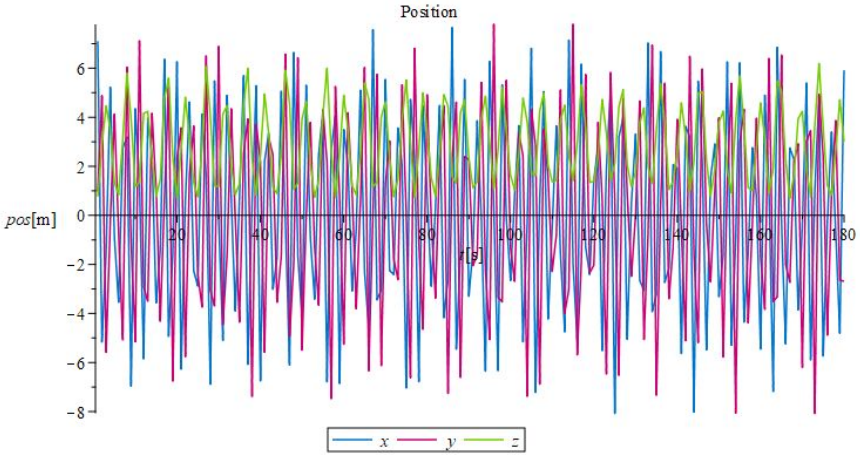
Global approach
(Numerical of 30 of 180 sec)

Position analysis of the seat from the ground

Recursive approach
Analytical

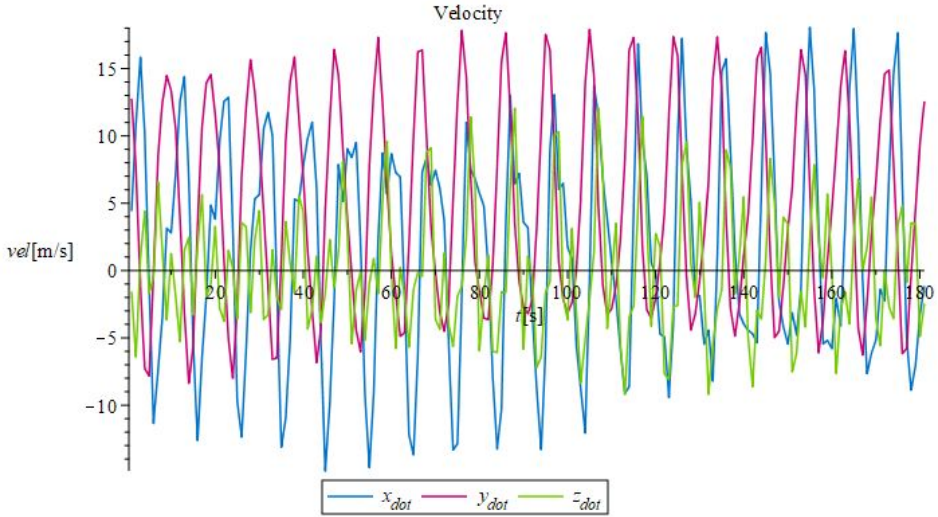


Global approach
Numerical

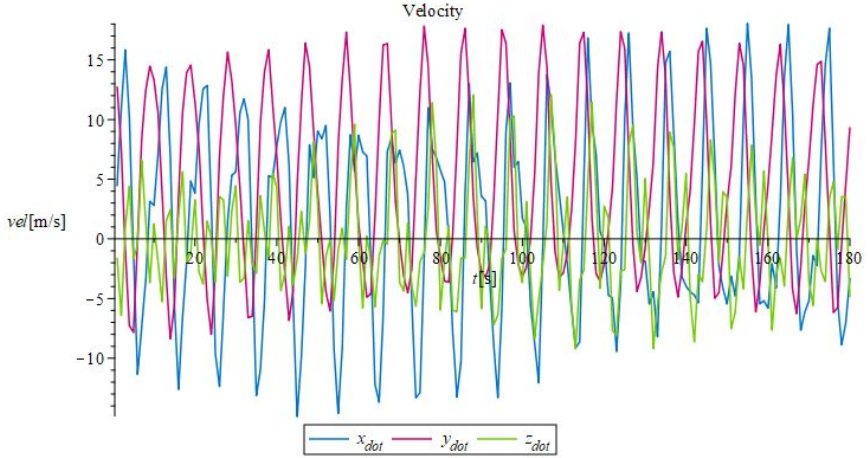


Velocity analysis of the seat

Recursive approach
Analytical

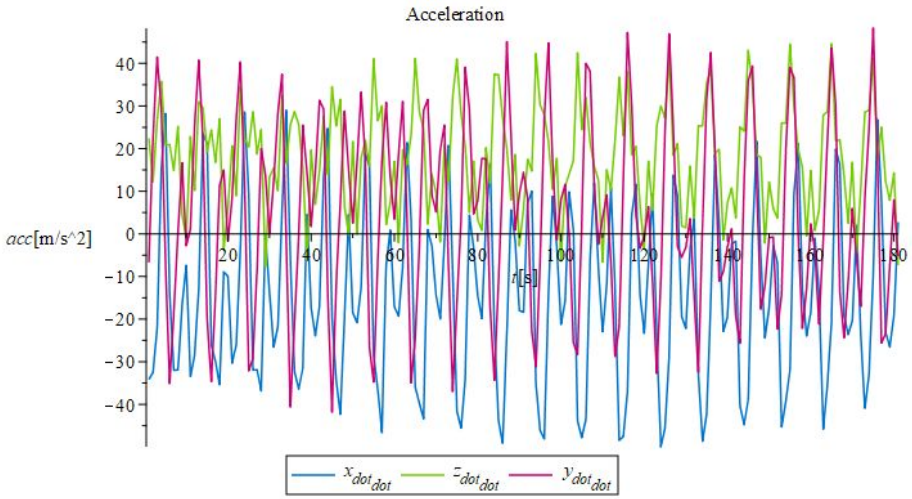


Global approach
Numerical

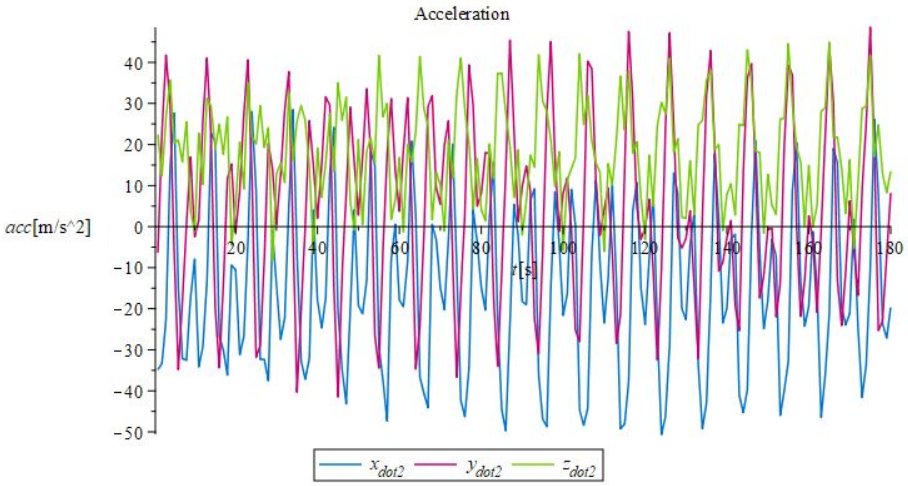


Acceleration analysis of the seat

Recursive approach
Analytical

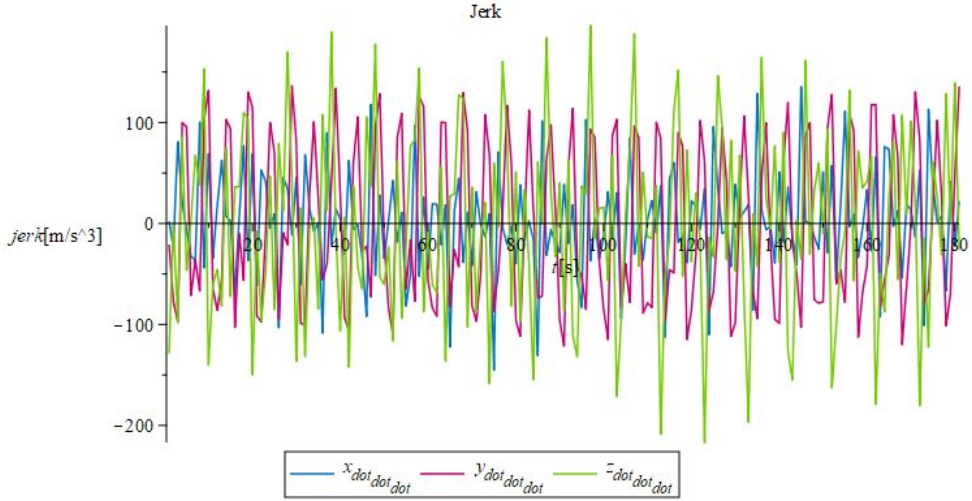


Global approach
Numerical

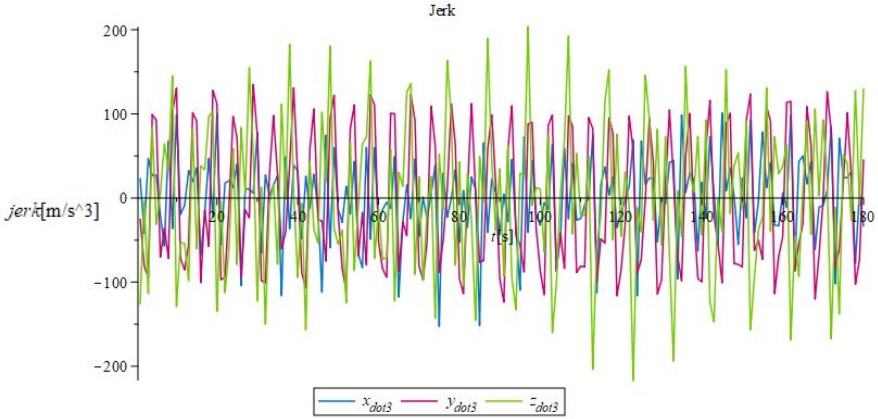


Jerks analysis of the seat

Recursive approach
(using splines)



Global approach
(using splines)



Fitness function

It mix three different function

- Durability: sum of jerks obtained;
- Safety: seconds where the acceleration of the seat is above the threshold following [Table1]
- Fun: seconds where the velocity of the seat it’s above 40 km/h

They are all mixed in one weighted sum

```
TotalFitness= -1*sec_vel_x_h+-1.2*sec_vel_x_l+-1.1*sec_vel_z_h+-1*sec_vel_z_l
              +1000*sec_not_safety
              +0.0001*avg_jerk
```

Montecarlo analysis

For each parameter of the motion profiles (rehonomous constraints) α_s, ω_s, ω_1, ω_5, ω_6 we gave a reasonable interval in which we operated the Monte Carlo Analysis. Such analysis consists in trying different values, chosen randomically in an interval, to determine which ones influence more a given function.

Standard deviation of the fitness function changing the parameters:

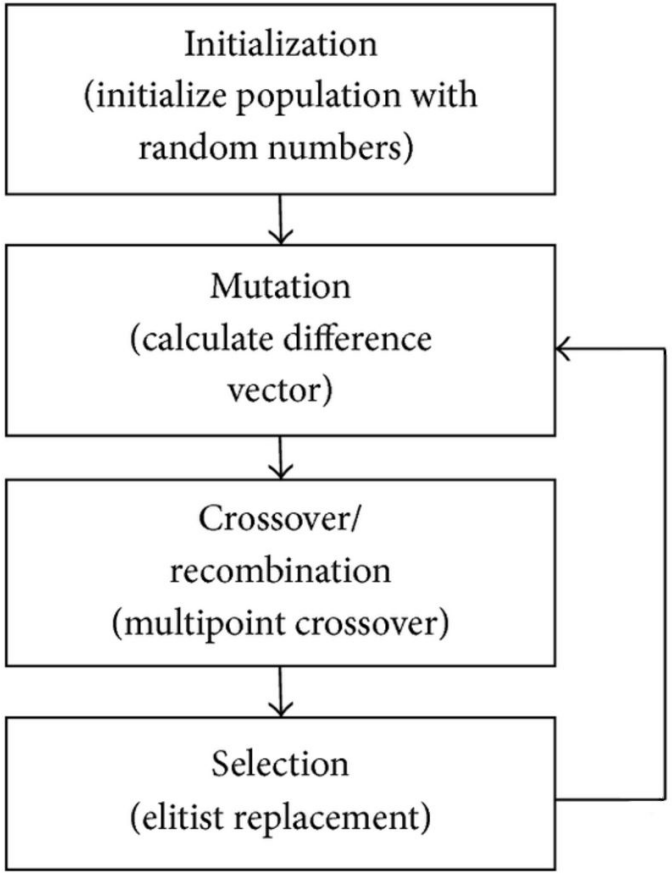
- α_s= 0.0017604450569581 3
- ω_1= 0.0009774203862414 34
- ω_5= 13937.1927683439
- ω_6= 634.670170016884
- ω_s= 17485.4200887781

ω_s is the parameter that influence more the performance indices.

Optimization

The parameters are optimized using Differential Evolution, a particular kind of Evolutionary Algorithms.
The objective was to minimize the fitness function.

Every individual of the population is composed by the parameter of the profiles (α_s, ω_s, ω_1, ω_5, ω_6).



Confrontation with target values

After the optimization we obtained the following:

Reached targets

- Velocity above 40 km/h or under -40 km/h along x for 36.2 seconds. The target was 30s
- Acceleration always within safety limits
- Dimensions within 20x20m

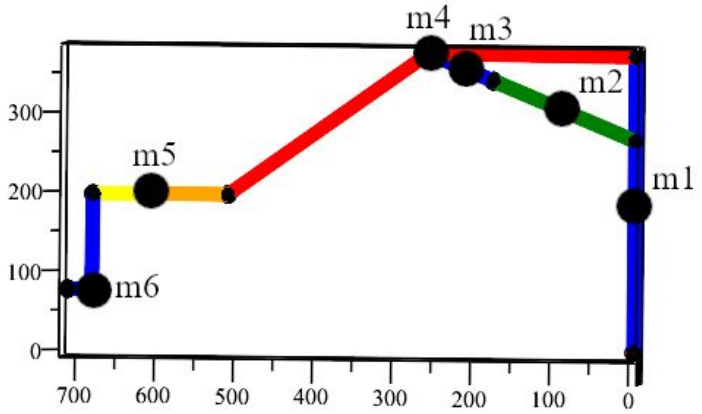
Missed targets

- Velocity above 40 km/h or under -40 km/h along z for 3.5 seconds. The target was 10s

Dynamic analysis

Weights estimate

We decided to develop a first approximation using point masses distributed as in the picture.




```
masses:={m1=300,m2=500,m3=500,m4=4000,
m5=750,m6=150}:
```

Hence we considered only one non-zero component of the inertia tensor for each mass:

```
inertias:={IZ1=((L2/2)^2)*m1, IX2=((L3/2)^2)*m3, IX3=((L4/2)^2)*m3,
IX4=((L5/2)^2)*m4, IX5=((L7/2)^2)*m5, IZ6=((L9/2)^2)*m6}
```

Dynamic analysis - Definition of the system

- First we defined masses and inertia tensors
- Then we defined the centers of mass and all the bodies
- After this, we defined the joints and relative forces and torques
- Finally, we used the command _gravity to add gravitational forces

Centers of Mass

```
> G1 := make_POINT(RF1_r, 0,0, L2/2):
  G2 := make_POINT(RF2_r, L3/2,0,0):
  G3 := make_POINT(RFp_3r, L4/2,0,0):
  G4 := make_POINT(RFp_4r, L5,0,0):
  G5 := make_POINT(RF5_r,L7,0,0):
  GSeat := make_POINT(RF6_r,0,0,-L9/2):
```

Definition of **bodies**

```
> B1 := make_BODY(G1, m1, 0,0,IZ1):
  B2 := make_BODY(G2, m2, IX2,0,0):
  B3 := make_BODY(G3, m3, IX3,0,0):
  B4 := make_BODY(G4, m4, IX4,0,0):
  B5 := make_BODY(G5, m5, IX5,0,0):
  BSeat := make_BODY(GSeat, m6, 0,0,IZ6): show(%):
```

Joints

```
> rev_1 := make_TORQUE(RF1_r,0,0,c1(t),B1):
  pris := make_FORCE(RFp_2r,prism(t),0,0,Pp_2r, B3, B2):
  rev_5 := make_TORQUE(RF5_r,0,0,c5(t),B5,B4):
  rev_Seat := make_TORQUE(RF6_r,0,c6(t),0,BSeat,B5):
```

We define the gravity and MBSymba will take into account gravitational forces

```
> _gravity := make_VECTOR(ground,0,0,-9.81):
```

Dynamic analysis with Lagrange approach

To develop the dynamic analysis of the system, we used the Lagrange approach.
To generate the Lagrange equations, we added the kinematic constraints and defined the MBS

LAGRANGE EQUATIONS

```
> lvars := [seq(lambda||i(t),i=1..nops(qd_vars))];
  constraints := make_CONSTRAINT(Phi, convert(lvars,list));
> Coords := convert(q_vars union lvars,list);
> mbs := {B1, B2, B3, B4, B5, BSeat,rev_1,pris,rev_5, rev_Seat, constraints};
```

Multibody system

```
> dae_sys := combine(simplify(lagrange_equations(mbs,Coords,t))):
```

DAE sytem of equations

```
> dae_vars1 , dae_sys1 := first_order(dae_sys, Coords, t): nops(dae_sys1);
```

Inverse Dynamic

Using positions, velocities and accelerations from kinematics (using a time-step of 1s/0.1s), we obtained forces and torques from the DAE

Define Arrays for solutions

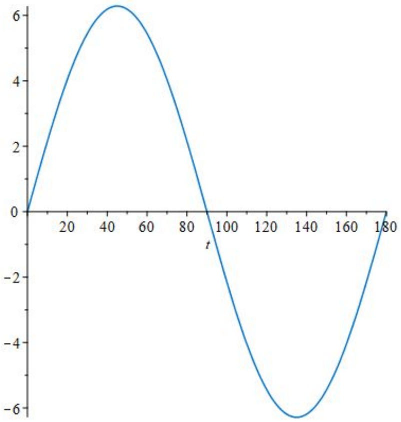
```
> pos := pos union pos_d;
  vel := diff(pos,t):
  acc := diff(vel, t):
  all_pos := Array(1 .. steps, []):
  all_vel := Array(1 .. steps, []):
  all_acc := Array(1 .. steps, []):
>
```

Obtain all the points

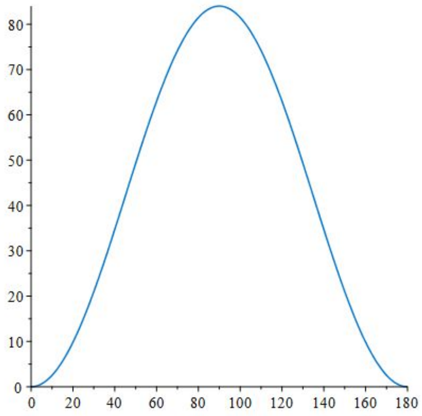
```
> for i from 1 by 1 to steps do
    if (irem(i,steps/18) = 0) then
       print(i)
    end if:
    nowPos:={};
    nowVel:={};
    nowAcc:={};
    for j from 1 to 6 do
      nowPos := nowPos union {lhs(pos[j])= evalf(subs(t=(i-1), data, rhs(pos[j])))};
      nowVel := nowVel union {lhs(vel[j])= evalf(subs(nowPos, t=(i-1), data, rhs(vel[j])))};
      nowAcc := nowAcc union {lhs(acc[j])= evalf(subs(nowVel, nowPos, t=(i-1), data, rhs(acc[j])))};
      #print(nowAcc);
    end do:
    all_pos[i]:= evalf(nowPos);
    all_vel[i]:= evalf(nowVel);
    all_acc[i]:= evalf(nowAcc);
  end do:

> motors_points := seq(fsolve(evalf(subs(vars_to_symbol,all_acc[i], all_vel[i],all_pos[i], inertias, data, masses,
  inv_dyn_eqns))[1..nops(q_vars)]),i = 1 .. steps):
```

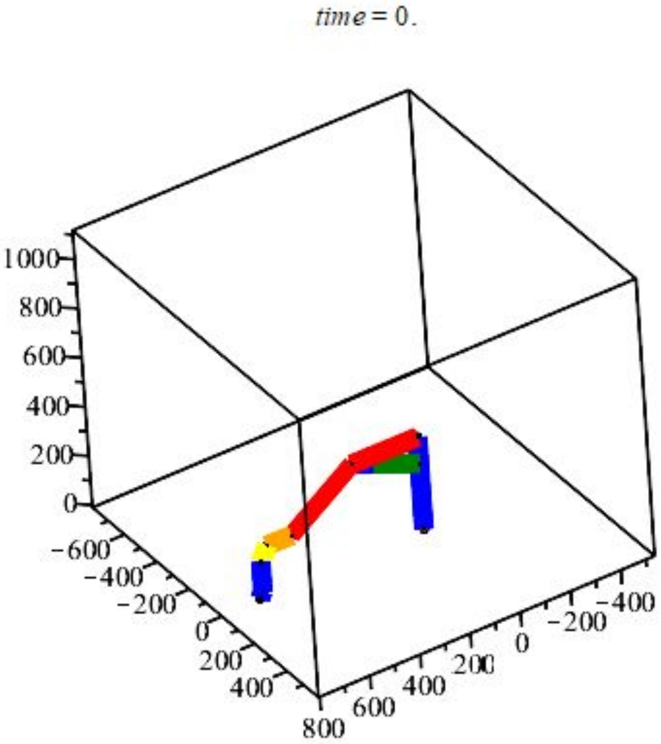
Simple profiles for Dynamics

Revolutes

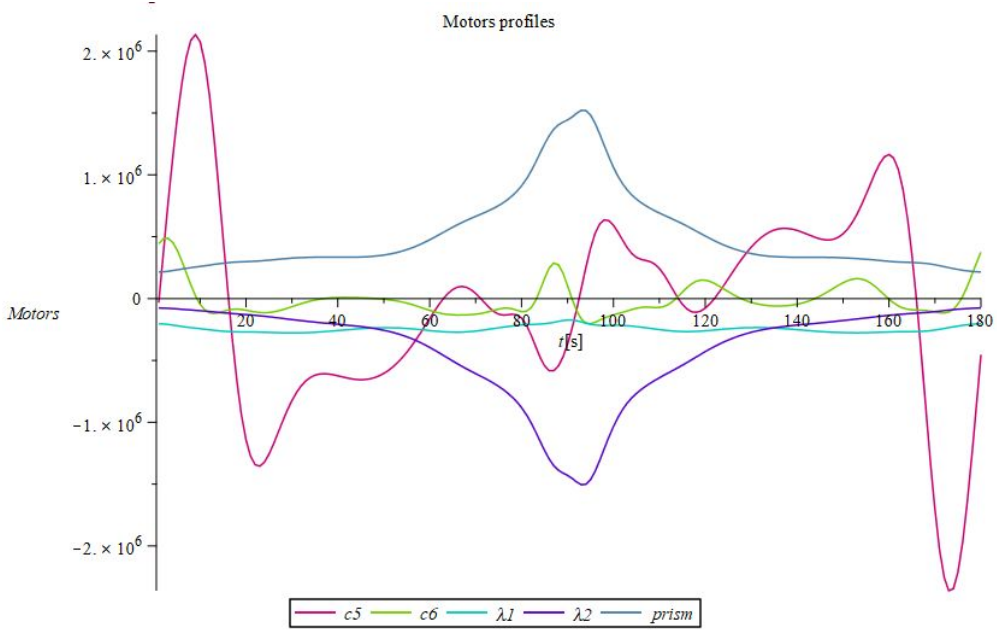


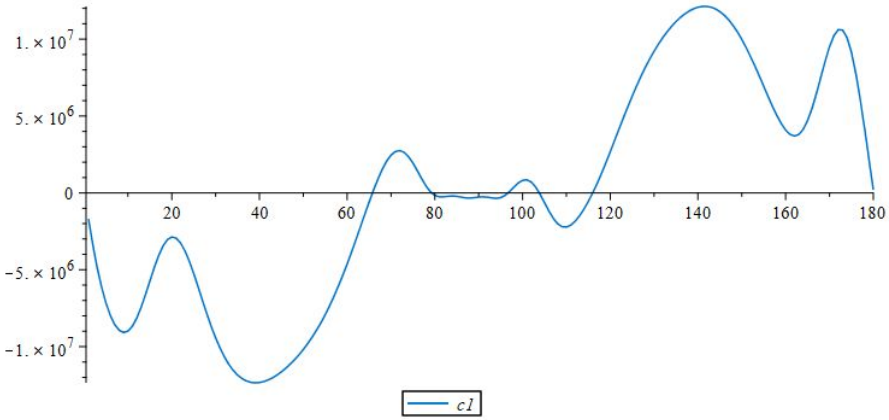
Prismatic





Motor Profiles





Initial Conditions

Starting from the motion profiles described before,
we computed the initial conditions using the DAE system

Initial conditions for reaction forces

```
> ICS := op(subs(t=0, all_pos[1]) union subs(t=0,down_order(all_vel[1],q_vars,t)));
> dae_sys[1..-nops(Phi)-1] union diff(Phi,t,t):

  AA, BB := GenerateMatrix(%,diff(q_vars,t,t) union l_vars):
  BB := down_order(BB,q_vars,t):
> LinearSolve(
  evalf(subs(t = 0, ICS, inertias, data, masses, AA)),
  evalf(subs(subs(c1=c1(t),c5=c5(t),c6=c6(t),prism = prism(t), motors_points[1]), t = 0, ICS, inertias, data, masses, BB)))
  ;
  ics := subs( t = 0, {ICS, seq(lvars[i]=%[i+nops(q_vars)],i=1..nops(lvars))});nops(%):
```

Numerical Solution

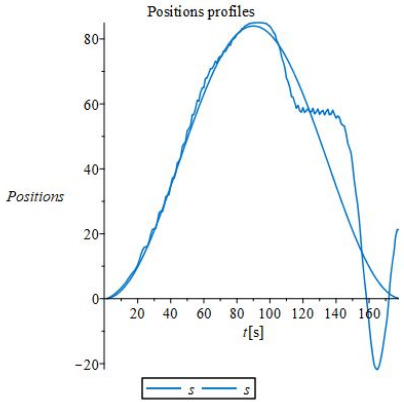
With the initial conditions and the values for the motors obtained before, we computed the solutions for the variables substituting them at each step using the following loop

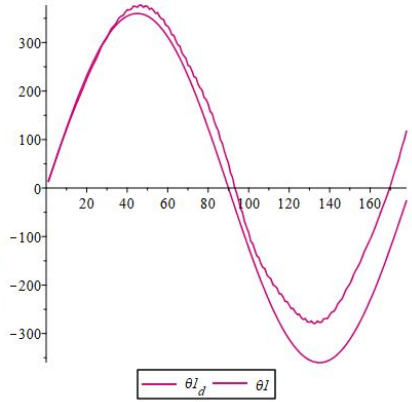
Numerical solution

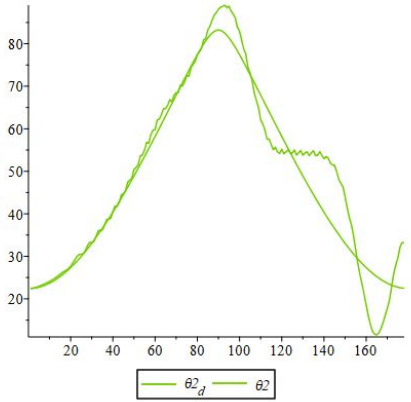
Loop for numerical solutions

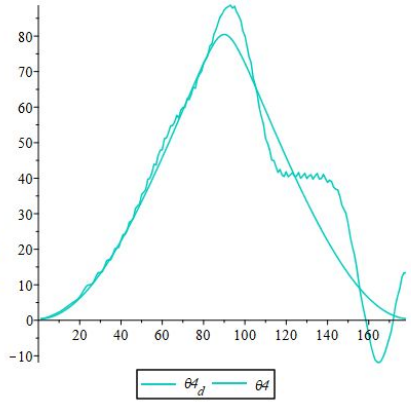
```
> motor_points_ics := subs(c1=c1(t),c5=c5(t), c6=c6(t),prism = prism(t),motors_points[1][1..3] union [motors_points[1][6]]):
  motor_points_ics := subs(c1=c1(t),c5=c5(t), c6=c6(t),prism = prism(t),motors_points[2][1..3] union [motors_points[2][6]]);
  dyn_sys := {op(subs(inertias, data, motor_points_ics, masses, dae_sys1))} union ics:

> sol := dsolve(dyn_sys, numeric, implicit=true);#, maxfun=10^5):

> Solutions := Array(1 .. steps + 1, []):
  Solutions[1] := sol(1)[2..-1]:
  #Solutions[2] := sol(0.1)[2..-1]:
  index_i := 0:
  for i from 1 to steps + 1 do # ADD 1 INSTEAD TO START FROM 0
    if (irem(i,10) = 0) then
       print(i);
    end if:
    motor_points_ics := subs(c1=c1(t),c5=c5(t),c6=c6(t),prism = prism(t),motors_points[i+2][1..3] union [motors_points[i+2][6]]):

    dyn_sys := {op(subs(inertias, data, motor_points_ics, masses, dae_sys1))} union subs(t=0,Solutions[i]):

    sol := dsolve(dyn_sys, numeric, implicit=true, maxfun=0):

    Solutions[i+1]:=sol(1)[2..-1]:

    index_i := i;
  end do:
```

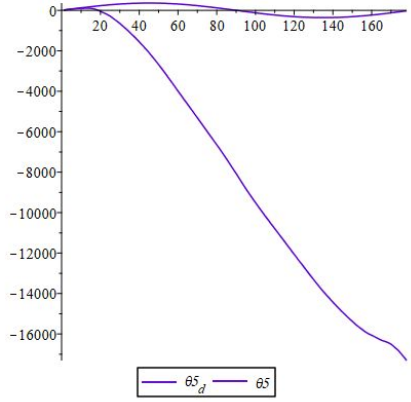
Plots (180 steps)

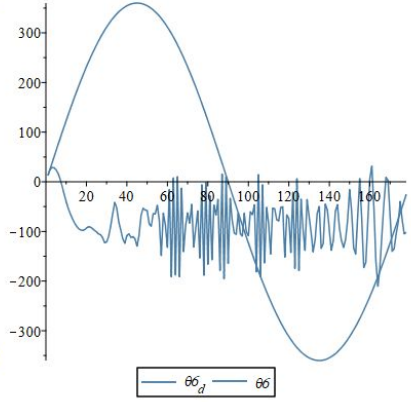


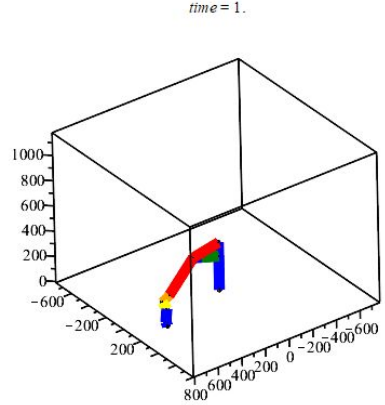




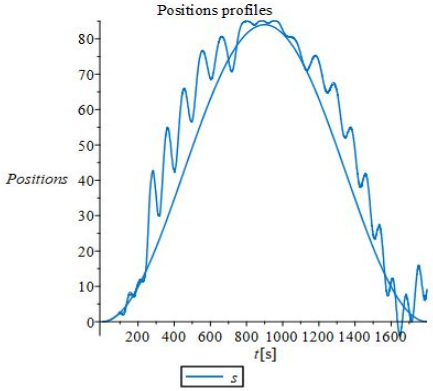


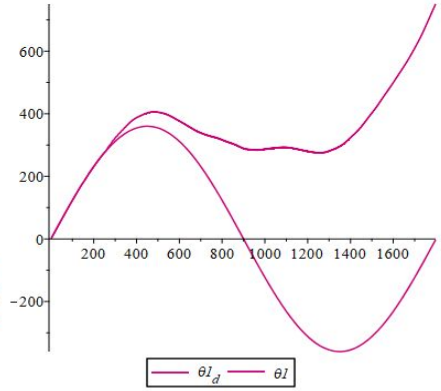


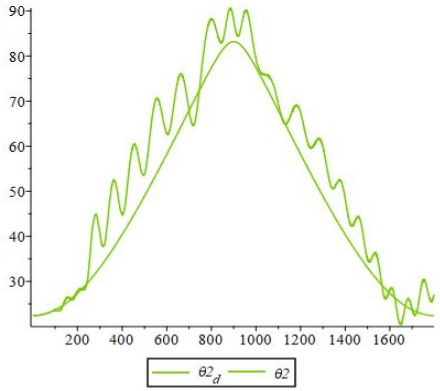


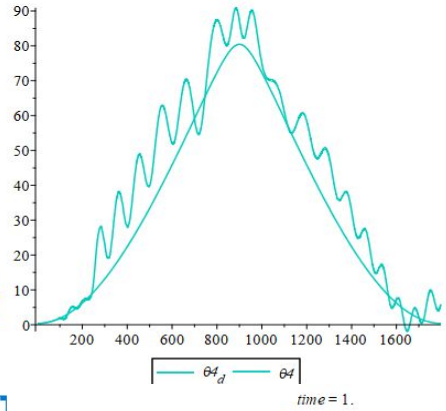


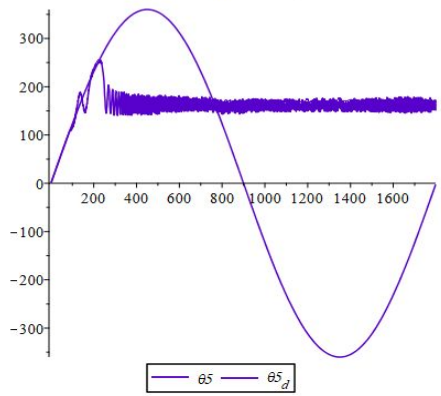
Plots (1800 steps)

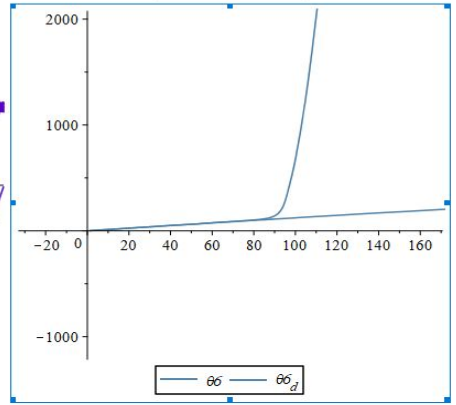


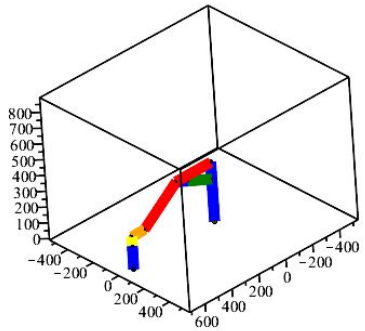






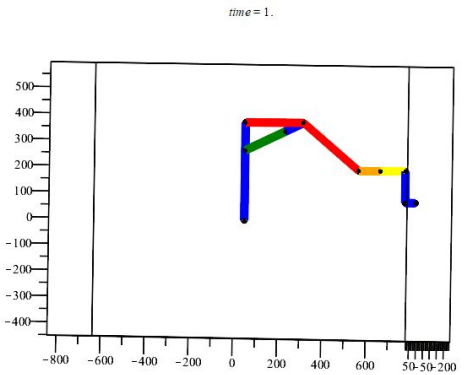


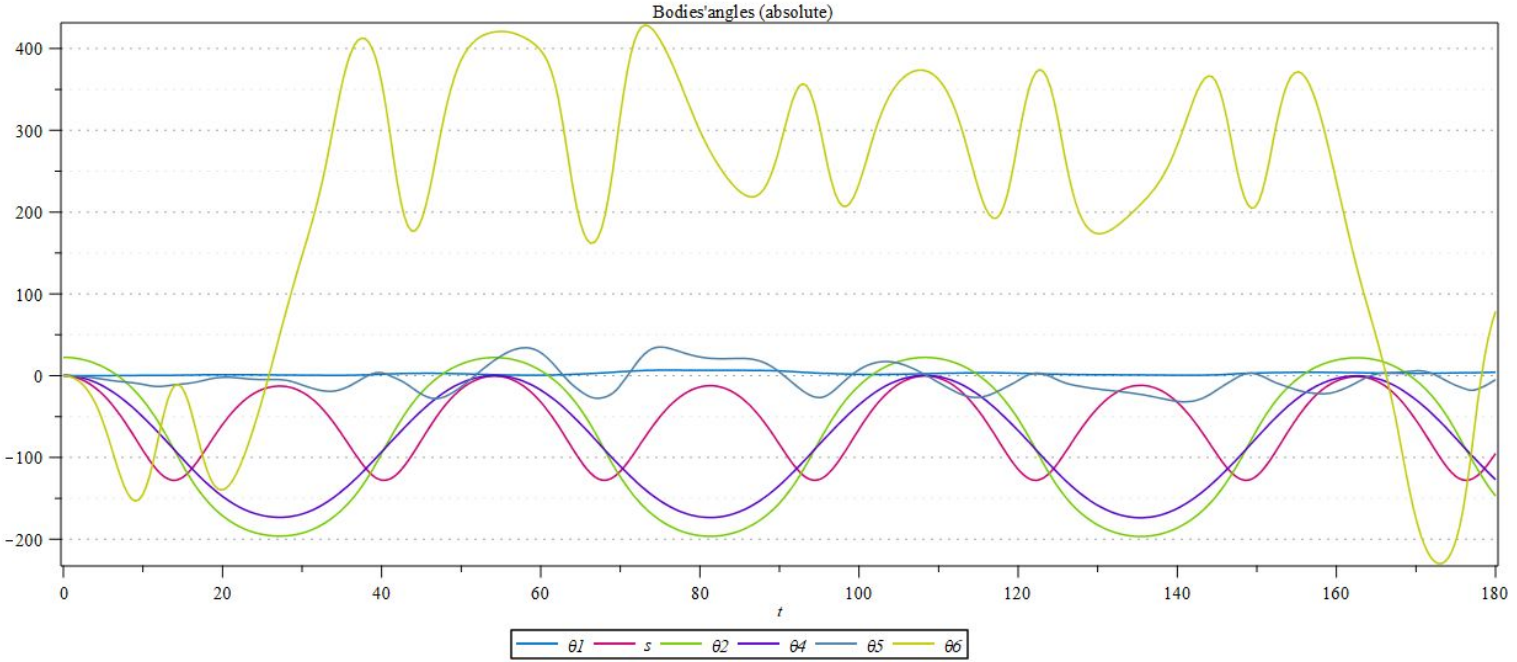




Static Dynamics

Imposing zero forces and torques we obtain:





Index Reduction

Using the library given by prof. Bertolazzi, we were able to reach the Index1 DAE, but the last step to obtain the ODE using Index reduction was not able to finish its execution

Index Reduction

```
> DAE1, ALG1 := DAEreduceBy1TheIndexBis( dae_sys1, diff(Coords,t) ):
  simplify(<DAE1>):
  simplify(<ALG1>):
> DAE2, ALG2 := DAEreduceBy1TheIndexBis( simplify(DAE1), diff(Coords,t) ):
  simplify(<DAE2>):
  simplify(<ALG2>):
> #DAE3, ALG3 := DAEreduceBy1TheIndexBis( simplify(DAE2), diff(Coords,t) ): #UnFeasible
```

Standard Approach

From DAE3 to ODE using Bertolazzi's Maple library latest version

Matrix form

```
> AA,BB := GenerateMatrix(dae_sys1,diff(dae_vars1,t)):
  Mass_maybe := AA[1..8,9..-1]:#Not sure it's the real mass matrix, to check
  simplify((AA . (Vector(diff(dae_vars1, t)))) - BB - Vector(dae_sys1)): #Evaluated to check
> POS := dae_vars1[1..6];
  VEL := dae_vars1[9..-3];
  ACC := [s__dot__dot(t), theta1__dot__dot(t), theta2__dot__dot(t), theta4__dot__dot(t), theta5__dot__dot(t),
  theta6__dot__dot(t), lambda1__dot__dot(t), lambda2__dot__dot(t)];
  MUL := lvars;
  Phi := Phi;
```

Extract mass

```
> Mass,RES := GenerateMatrix(dae_sys1,diff(VEL,t)):Mass:=Mass[1..-9]:
```

Extract Phi_q^t

```
> Phi_q_T,res := GenerateMatrix(dae_sys1,{lambda1(t),lambda2(t)}):DAE_3_Phi_q_t:
  Phi_q:=simplify(Transpose(Phi_q_T[1..6])):
> Phi_q_T, res := GenerateMatrix(convert(-RES, list), {lambda1(t), lambda2(t)}):
  Phi_q := simplify(Transpose(Phi_q_T[1 .. 6])):
```

Extract forces

```
> res[-8 .. -1]:
> gen_forces := res[1 .. -9]:
  nops(POS);
  nops(VEL);
  nops(Mass[1]);
> A, B, Phi_Q, H, W := DAEindex3ToODE(Mass, Phi, convert(gen_forces, list), POS, VEL):
```

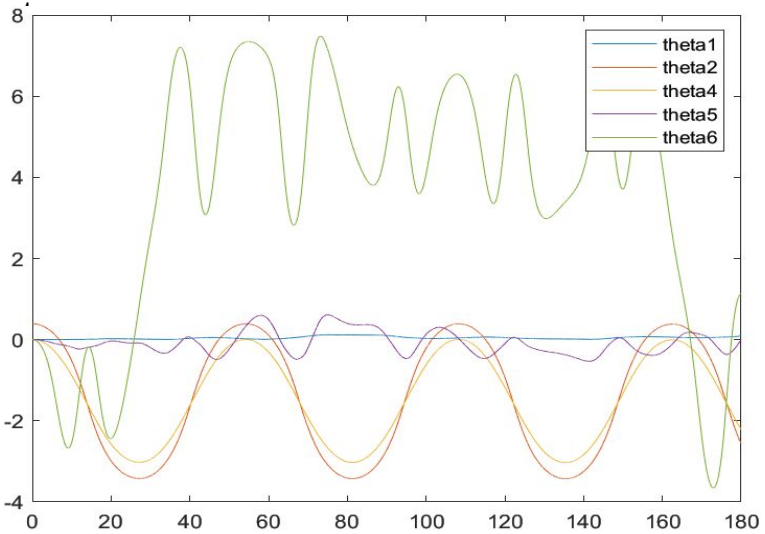
▶ *Matlab Code Generation*

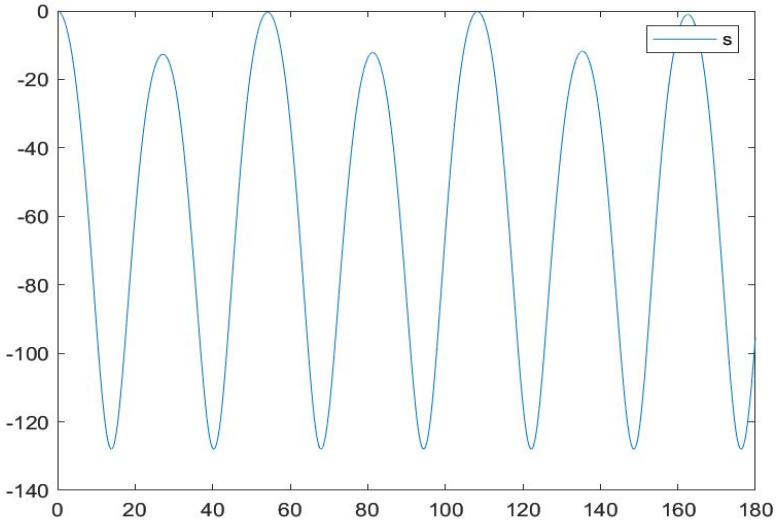
Resolution of ODE in Matlab with Projection

To implement the Winx(`Winx.m`) we used the `ODEbaseClass_P` from the toolbox of prof. Bertolazzi.

All the matrices are created using `DAEindex3ToODE` and exported using the functions from `DAE-lib.maplet`.

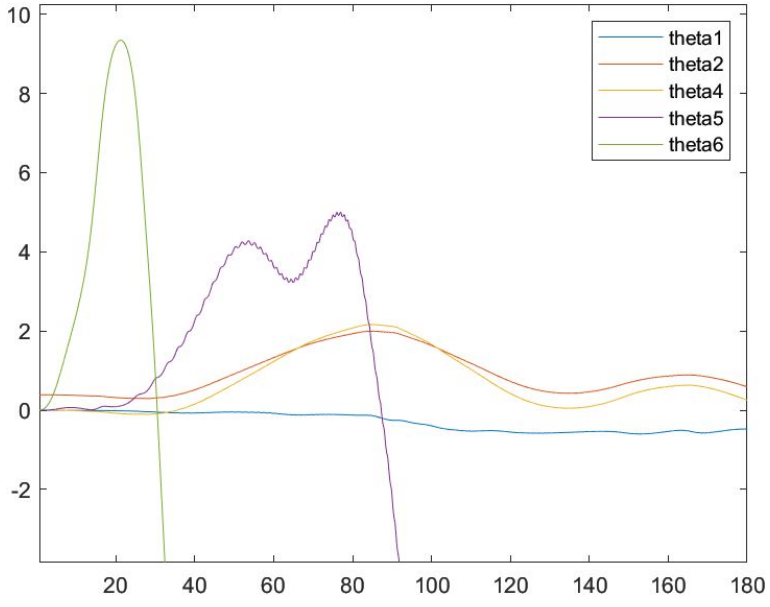
Coordinates evaluated using Heun numerical method with projection and all torques and forces set

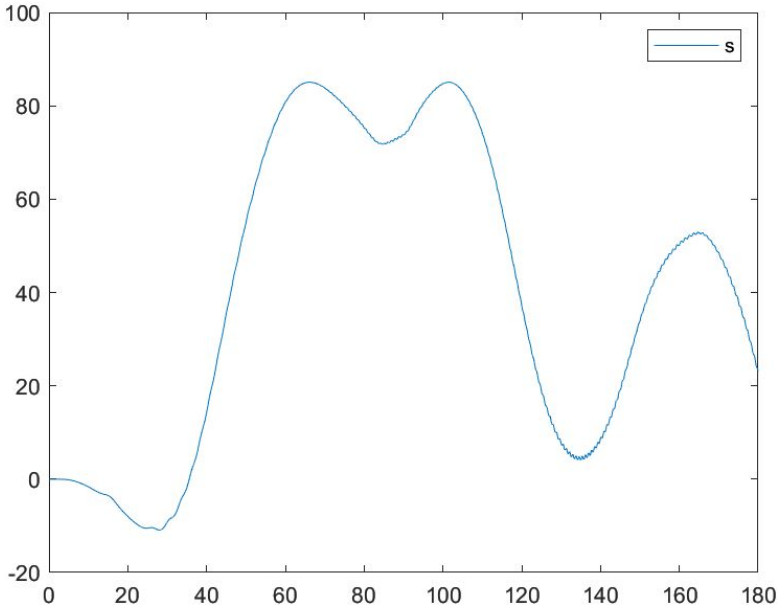




Resolution of ODE in Matlab with Projection

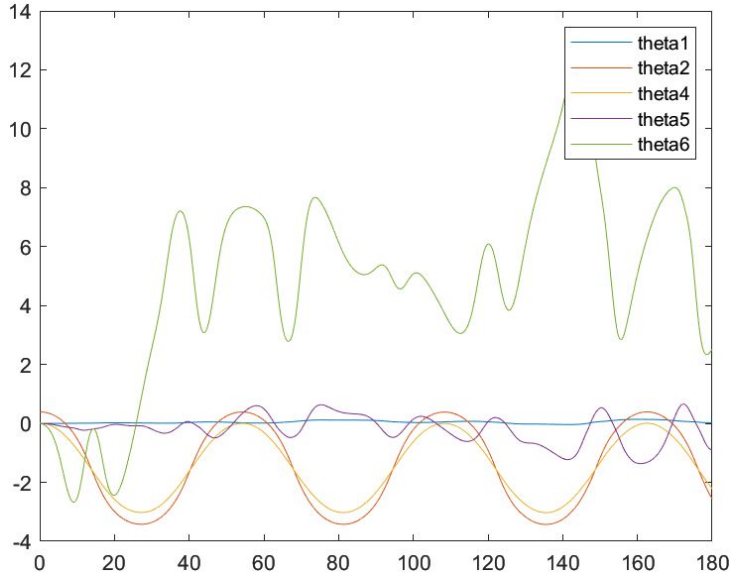
Coordinates evaluated using Heun numerical method with projection using motor profiles obtained from inverse dynamic

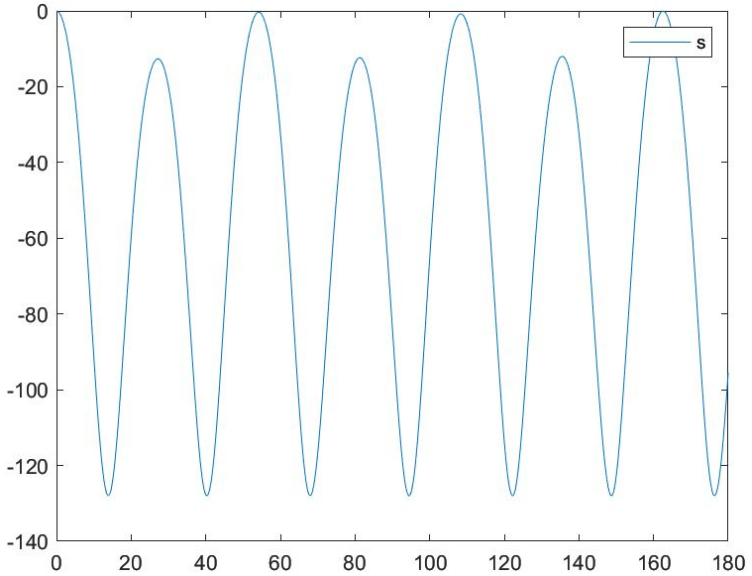




Resolution of DAE in Matlab with Baumgarte

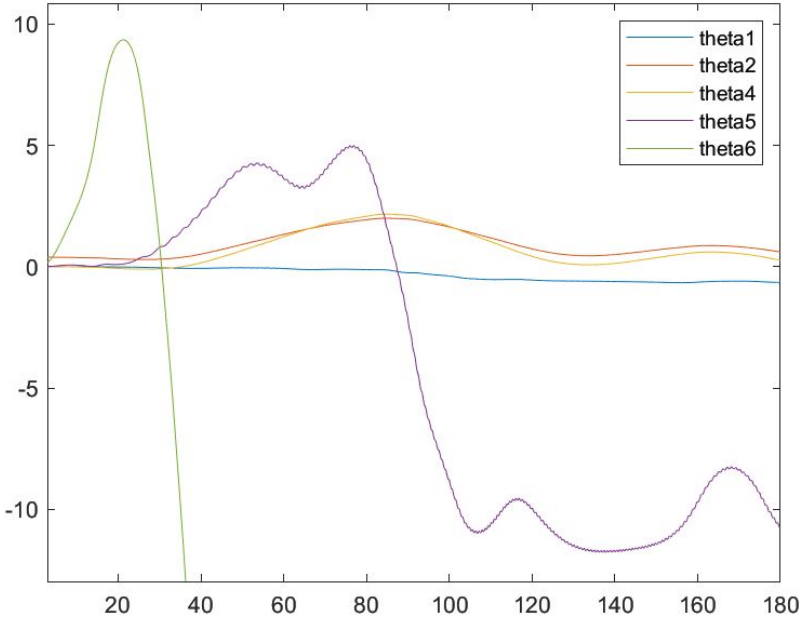
To implement the Winx (`WinxDAE.m`) we used the `DAE3baseClassImplicit` that represents a DAE equation and Baumgarte stabilization with η = 0.5 and ω= 4.
All torques and the prismatic force are fixed to 0.

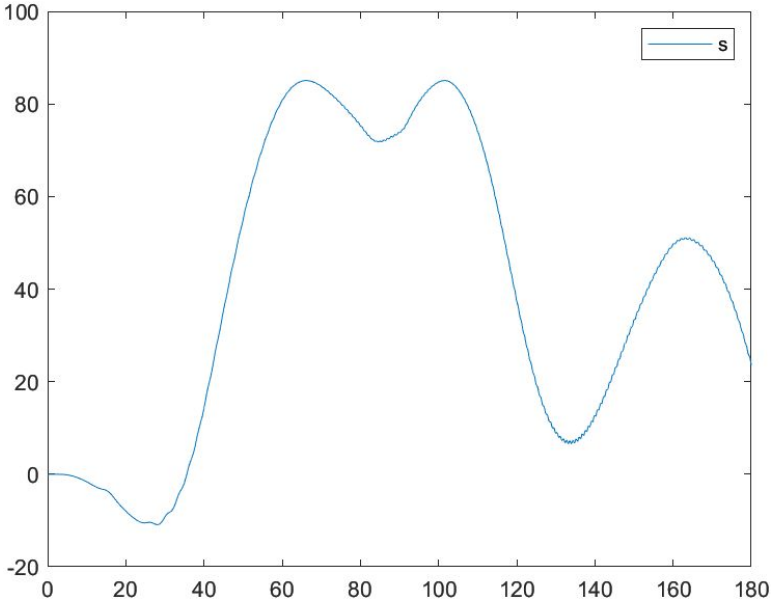




Resolution of DAE in Matlab with Baumgarte

Profile obtained by using Winx(`WinxDAE.m`) that represent a DAE equation and Baumgarte stabilization with η = 0.5 and ω= 4.
The motor profiles used are the ones obtained from the inverse dynamic.

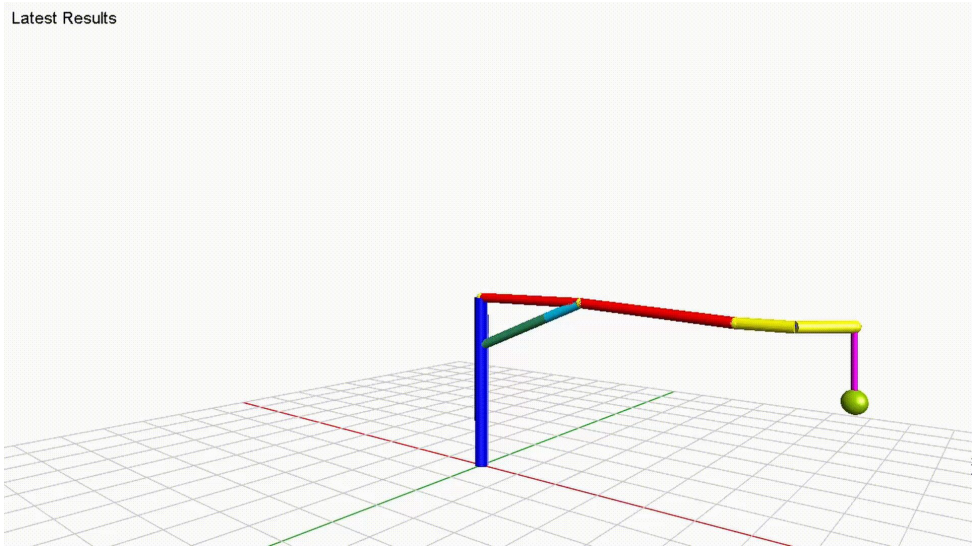




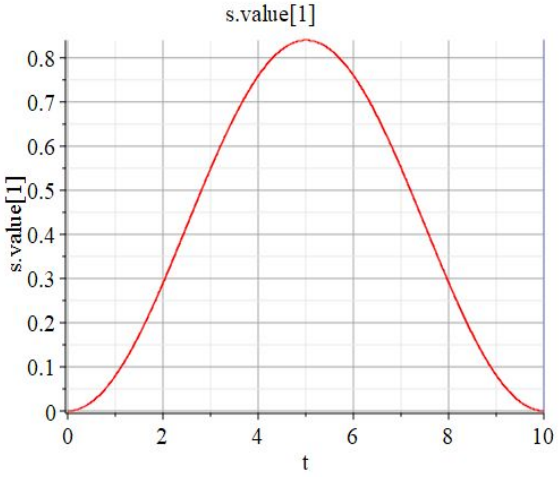
MapleSim Model

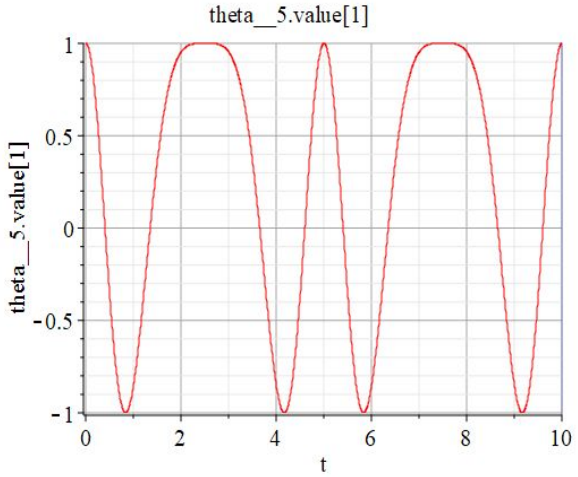
We recreated the model in MapleSim using the same geometries, masses and inertias considered for the model in Maple.

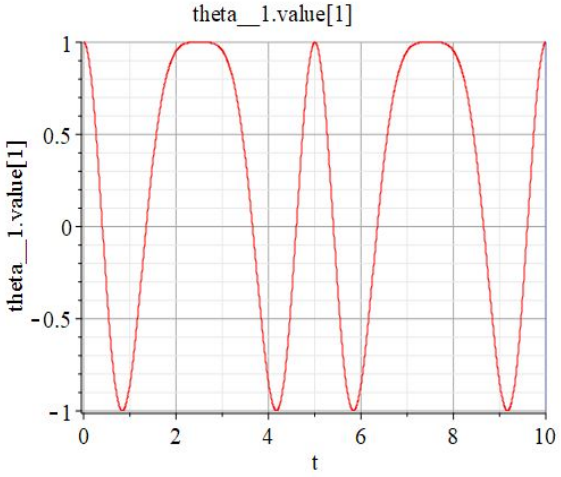
The motion profiles imposed at the joints are the same imposed when calculating the forces and torques in the dynamic analysis.

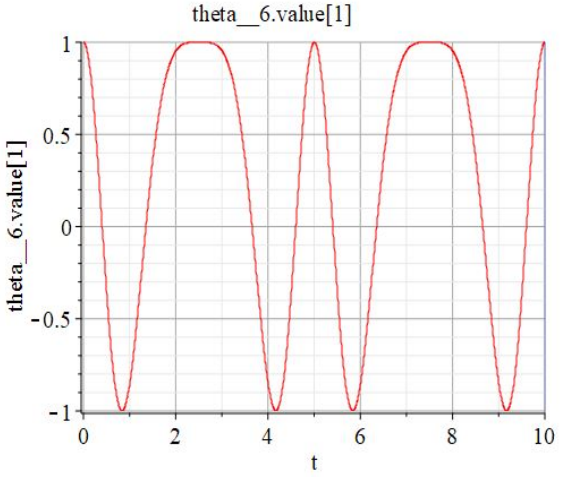


MapleSim Model - Joints motion

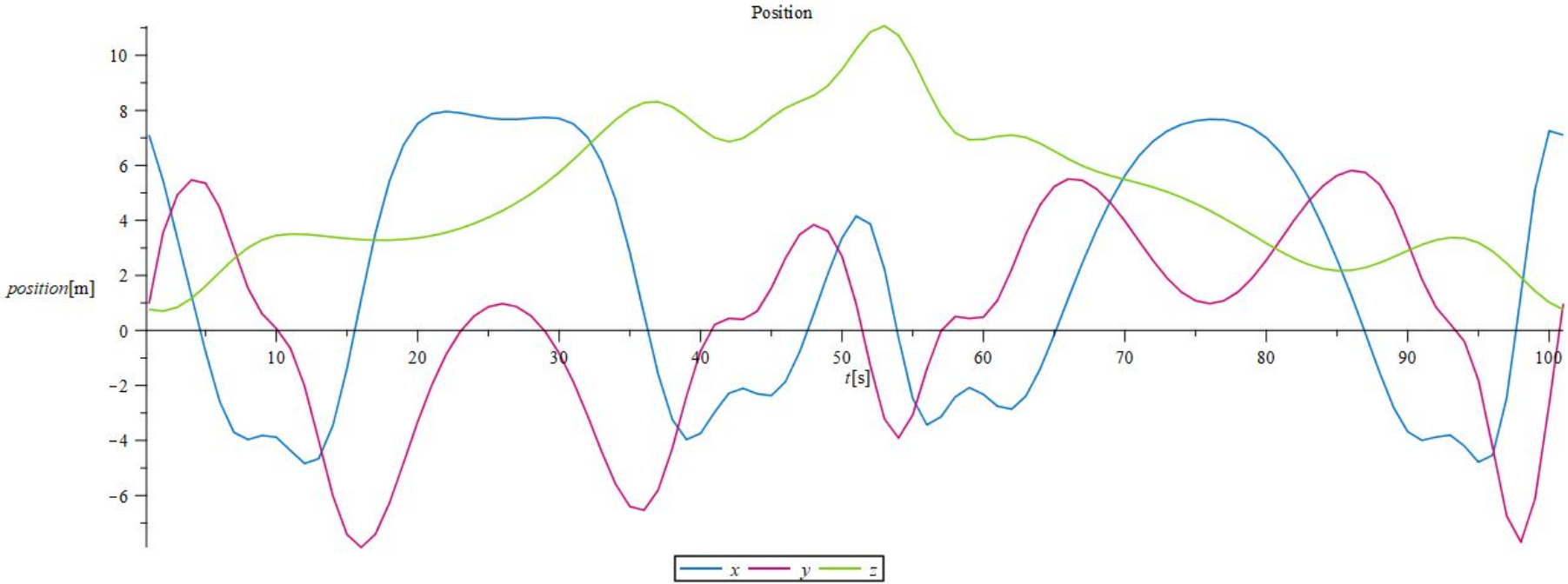




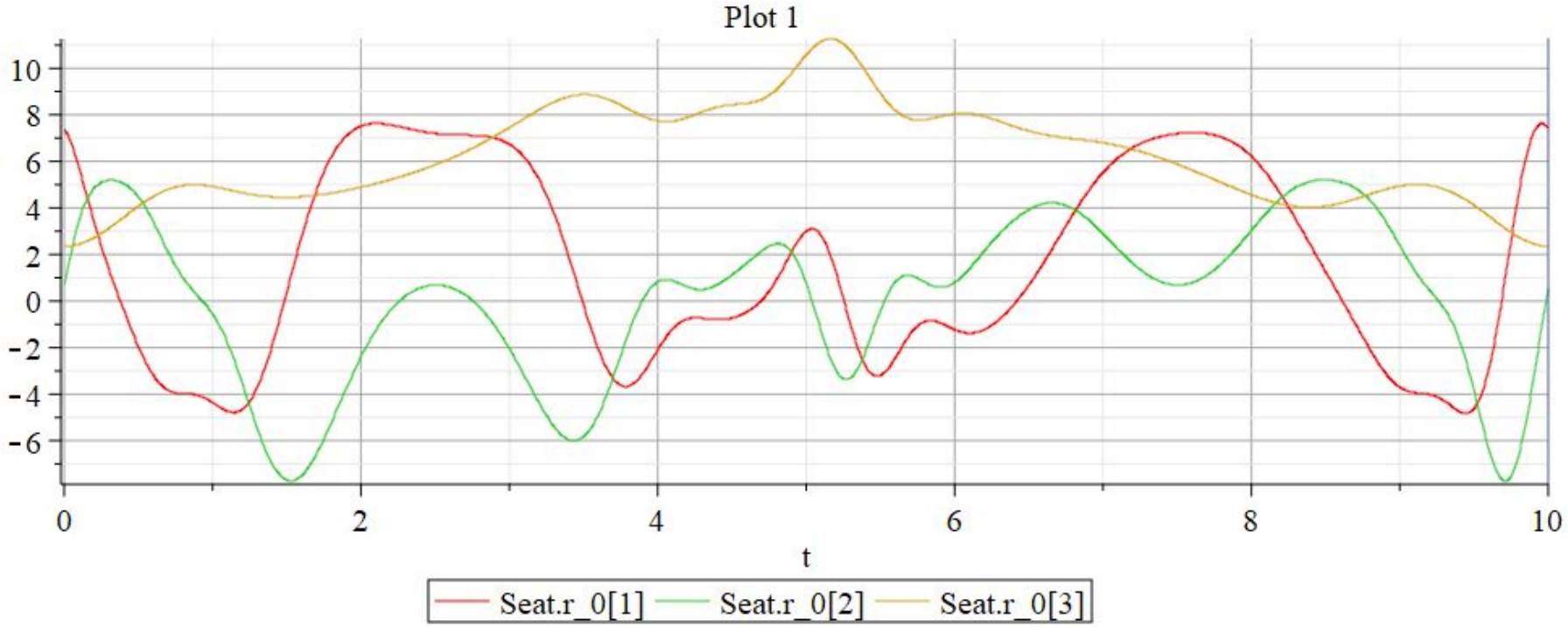




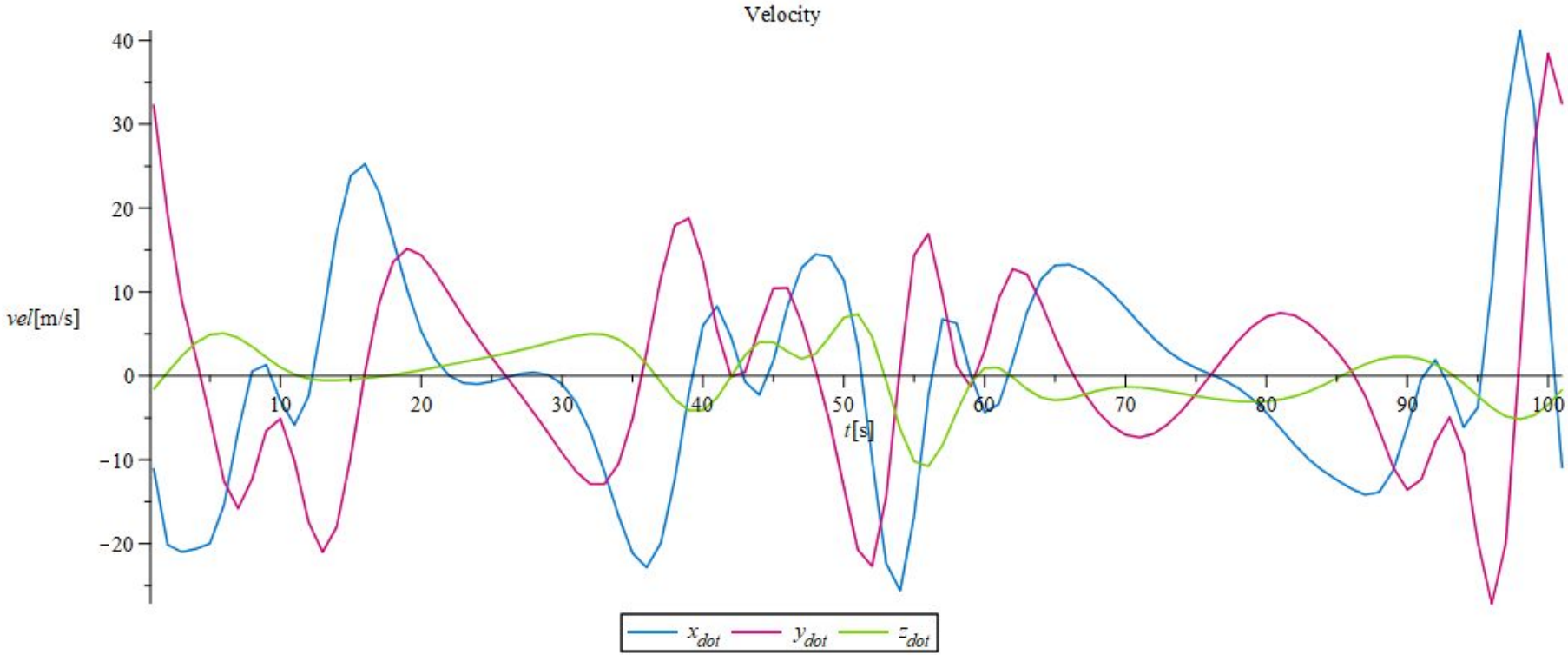
Desired position plots



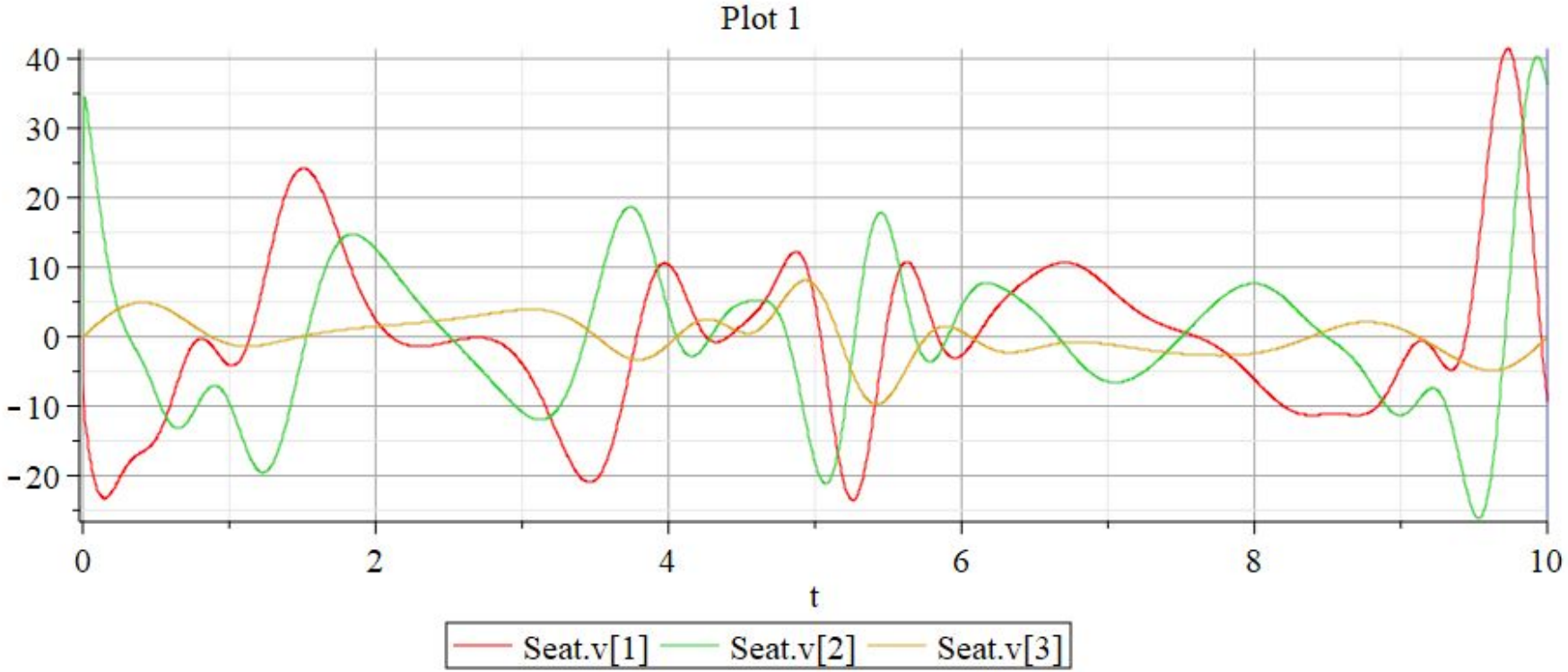
MapleSim Model - position plots



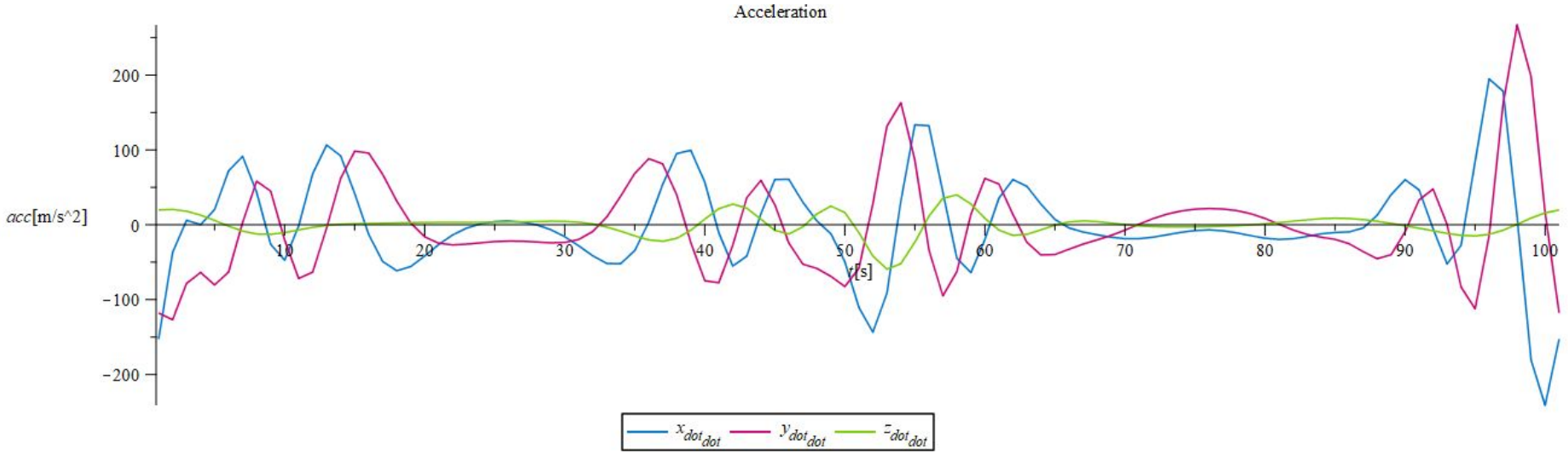
Desired velocity plots



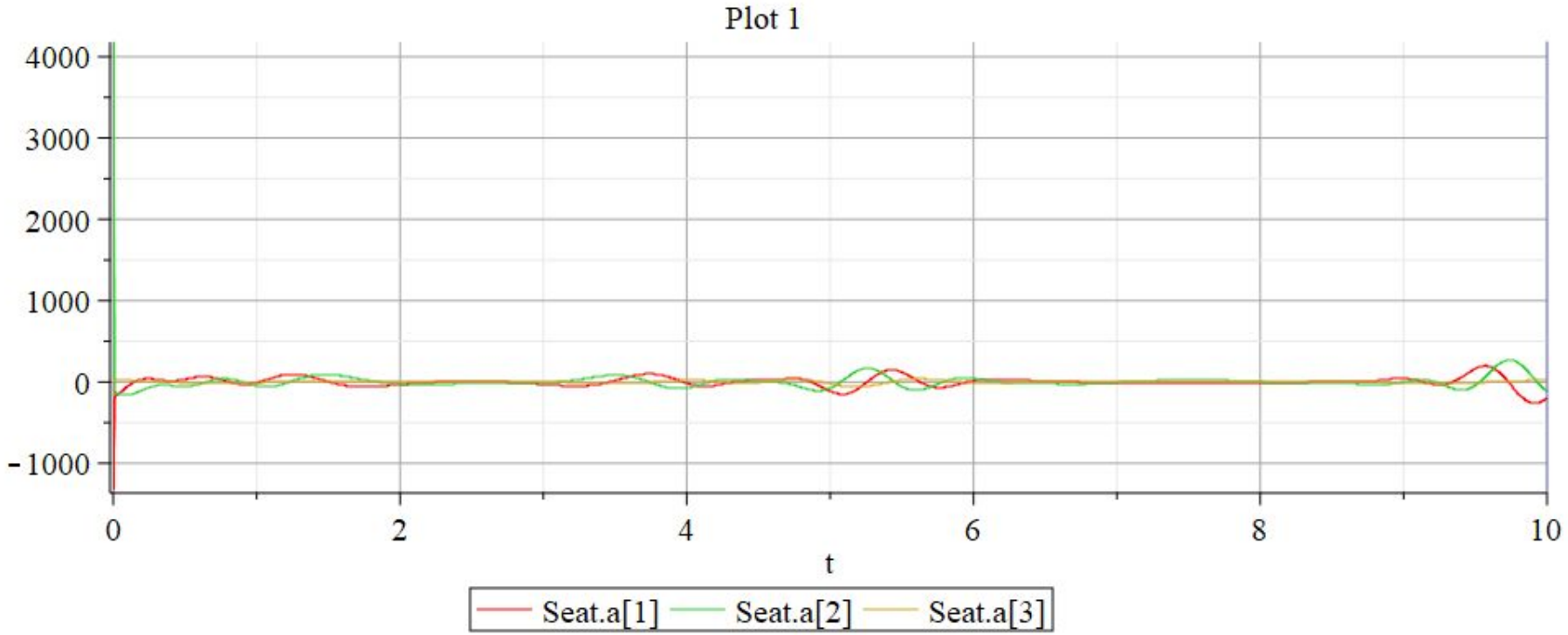
MapleSim Model - velocity plots



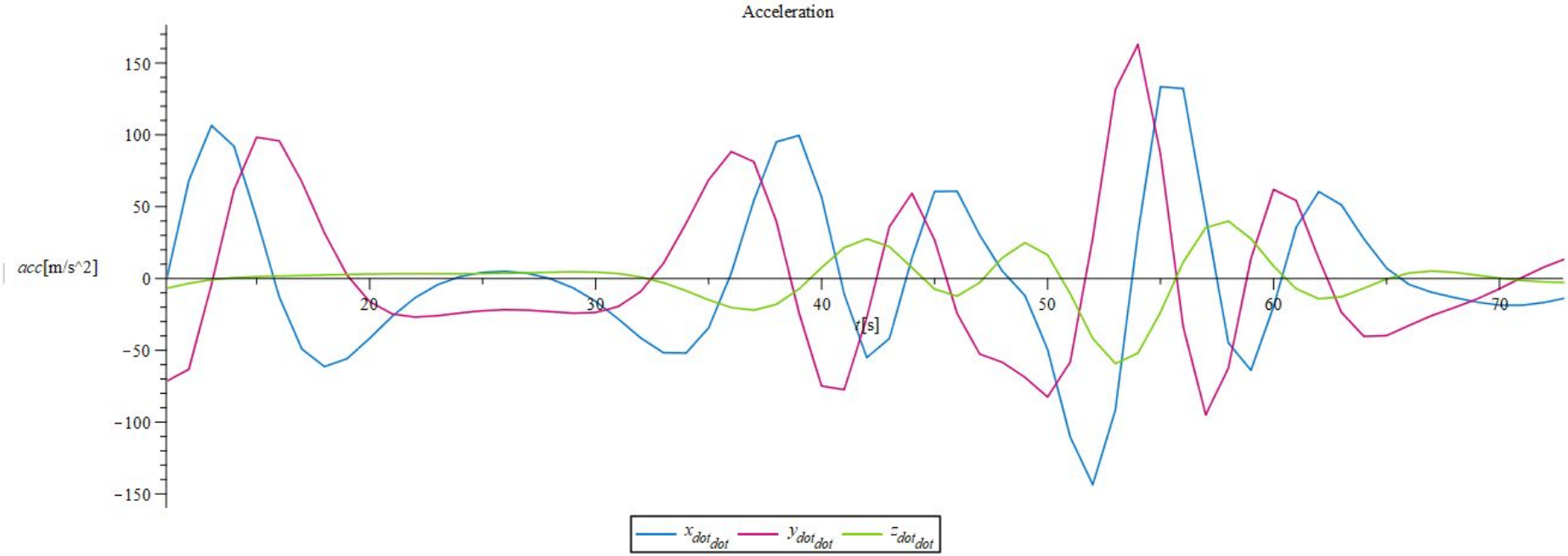
Desired acceleration plots



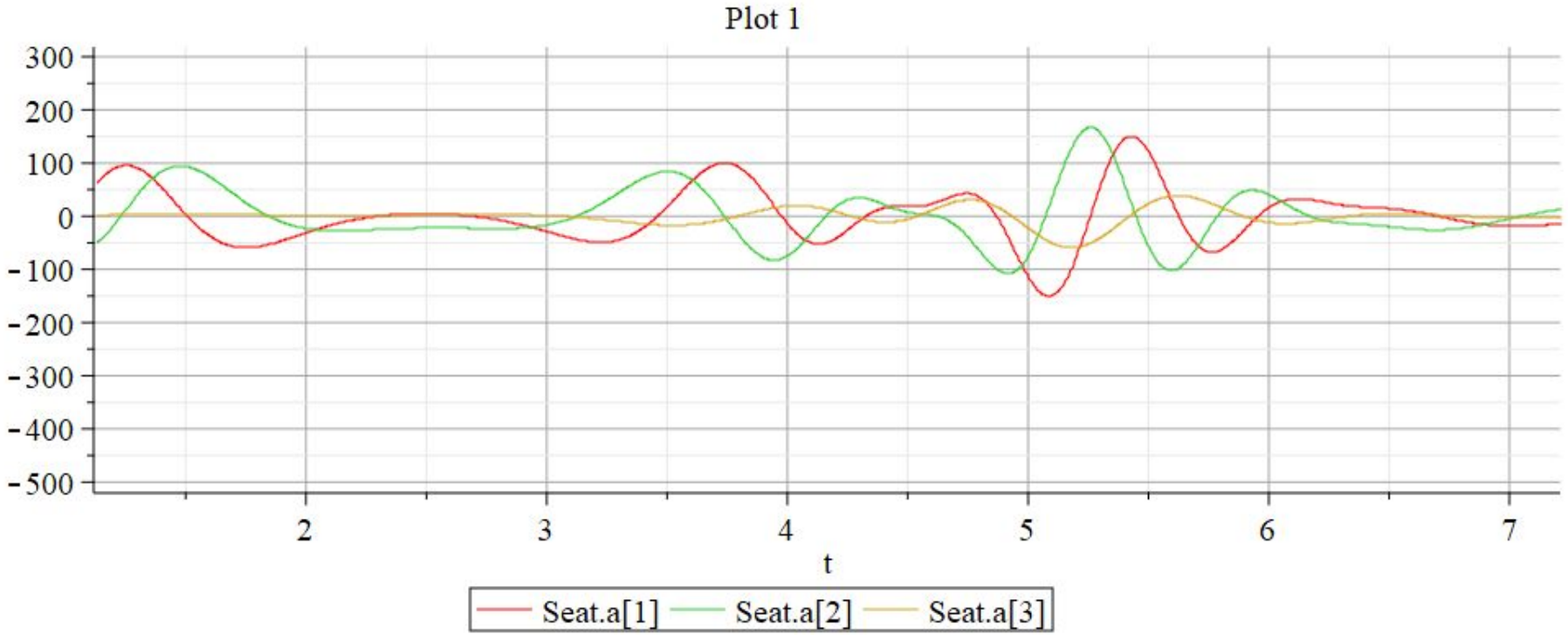
MapleSim Model - acceleration plots



Desired acceleration plots

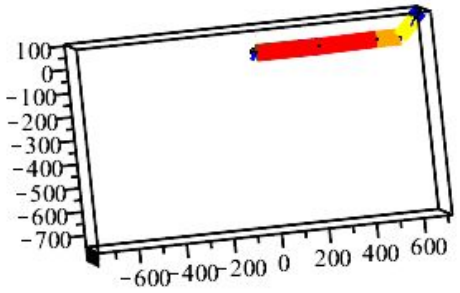


MapleSim Model - acceleration plots



MapleSim Model - DAE

Gravity on Y



Gravity on Z

```
> sol := dsolve(dyn_sys, numeric, implicit=true);
Error, (in dsolve/numeric/DAE/implicit) invalid
subscript selector
```

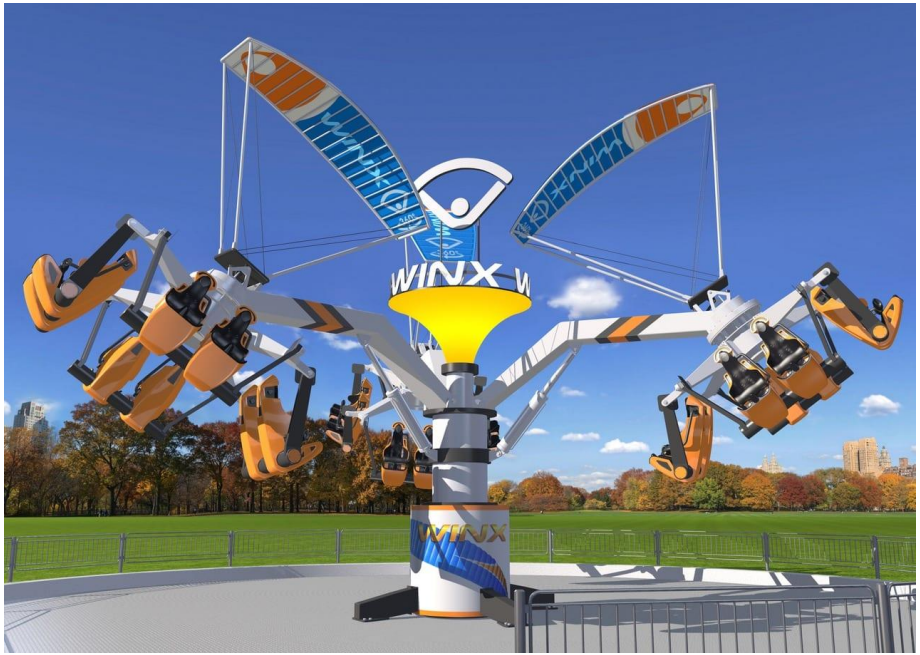
Limits and problems

Limits of our simulation:

- The masses are considered as concentrated in the center of mass, and consequently also the inertias are simplified
- We did not consider friction in our model
- We just considered 1 arm and 1 seat

Problems we have:

- The results of the inverse dynamic diverge from the profile we expected
- We did not manage to solve the DAEs exported from the MapleSim model (with the correct gravity configuration)
- Matlab solution obtained using the motor profiles of the joints even if we get similar results for both methods (projection and Baumgarte)



Thank you

All the code is available at https://github.com/MesSem/Modeling2020