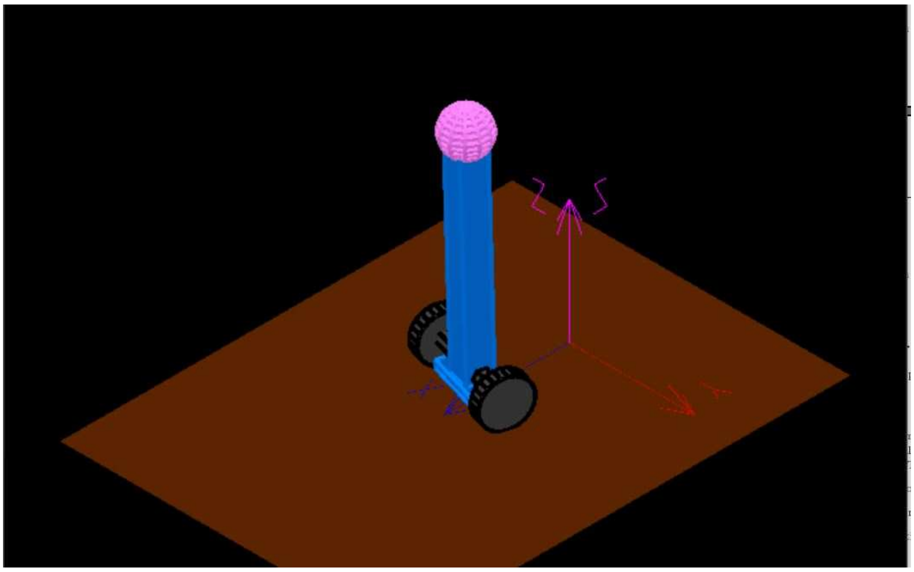
Segway Tracking Control

April 29, 2021



1 General data of the studied mechanism

The system comprises 6 bodies (defined by the global variable `nbrbody`). Each body is called S_j (j from 0 to 5). The number of degrees of freedom of the system is 8 (`nbrdof`). The configuration parameters are denoted by q_i (i from 0 to 7).

The inertial data, given by the user, consist of the mass m_{Si} and the inertia tensor $\Phi_{G,Si}$ of each body i expressed with respect to the center of gravity.

$$m_{S0} = 1.0 \cdot 10^{-10}\, kg$$

$$m_{S1} = 90\, kg$$

$$m_{S2} = 15\, kg$$

$$m_{S3} = 15\, kg$$

$$m_{S4} = 1\, kg$$

$$m_{S5} = 1\, kg$$

$$\Phi_{G,S0} = \begin{bmatrix} 1.0 \cdot 10^{-10} & 0 & 0 \\ 0 & 1.0 \cdot 10^{-10} & 0 \\ 0 & 0 & 1.0 \cdot 10^{-10} \end{bmatrix} \quad , \text{ en } kg.m^2$$

$$\Phi_{G,S1} = \begin{bmatrix} 21.6 & 0 & 0 \\ 0 & 21.6 & 0 \\ 0 & 0 & 2.5 \end{bmatrix} \quad \text{, en } kg.m^2$$

$$\Phi_{G,S2} = \begin{bmatrix} 0.2 & 0 & 0 \\ 0 & 0.3 & 0 \\ 0 & 0 & 0.2 \end{bmatrix} \quad \text{, en } kg.m^2$$

$$\Phi_{G,S3} = \begin{bmatrix} 0.2 & 0 & 0 \\ 0 & 0.3 & 0 \\ 0 & 0 & 0.2 \end{bmatrix} \quad \text{, en } kg.m^2$$

$$\Phi_{G,S4} = \begin{bmatrix} 0.00012 & 0 & 0 \\ 0 & 0.0001 & 0 \\ 0 & 0 & 0.00012 \end{bmatrix} \quad \text{, en } kg.m^2$$

$$\Phi_{G,S5} = \begin{bmatrix} 0.00012 & 0 & 0 \\ 0 & 0.0001 & 0 \\ 0 & 0 & 0.00012 \end{bmatrix} \quad \text{, en } kg.m^2$$

2 Complete kinematics calculated by Sympy

The following parameters have been calculated from the user's file `segway.py` with a CPU time of 3.322 second(s).

Relative motions

The motion of some bodies has been defined as a relative motion with respect to another body. It is the case of 4 bodies, for which the motion is defined in the following manner :

- Motion of body S_2 is given with respect to body S_1.
- Motion of body S_3 is given with respect to body S_1.
- Motion of body S_4 is given with respect to body S_1.
- Motion of body S_5 is given with respect to body S_1.

Homogeneous transformation matrix of each body

$$T_{0G,S0} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{0G,S1} = \begin{bmatrix} \cos(q_4)\cos(q_5) & -\sin(q_5)\cos(q_4) & \sin(q_4) \\ \sin(q_3)\sin(q_4)\cos(q_5) + \sin(q_5)\cos(q_3) & -\sin(q_3)\sin(q_4)\sin(q_5) + \cos(q_3)\cos(q_5) & -\sin(q_3)\cos(\\ \sin(q_3)\sin(q_5) - \sin(q_4)\cos(q_3)\cos(q_5) & \sin(q_3)\cos(q_5) + \sin(q_4)\sin(q_5)\cos(q_3) & \cos(q_3)\cos(q \\ 0 & 0 & 0 \end{bmatrix}$$

$$T_{refG,S2/S1} = \begin{bmatrix} \cos(q_6) & 0 & \sin(q_6) & 0 \\ 0 & 1 & 0 & -0.4 \\ -\sin(q_6) & 0 & \cos(q_6) & -0.9 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{refG,S3/S1} = \begin{bmatrix} \cos(q_7) & 0 & \sin(q_7) & 0 \\ 0 & 1 & 0 & 0.4 \\ -\sin(q_7) & 0 & \cos(q_7) & -0.9 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{refG,S4/S1} = \begin{bmatrix} \cos(25q_4 - 25q_6) & 0 & -\sin(25q_4 - 25q_6) & 0 \\ 0 & 1 & 0 & -0.25 \\ \sin(25q_4 - 25q_6) & 0 & \cos(25q_4 - 25q_6) & -0.9 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{refG,S5/S1} = \begin{bmatrix} \cos(25q_4 - 25q_7) & 0 & -\sin(25q_4 - 25q_7) & 0 \\ 0 & 1 & 0 & 0.25 \\ \sin(25q_4 - 25q_7) & 0 & \cos(25q_4 - 25q_7) & -0.9 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Their time derivative

$$\dot{T}_{0G,S0} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\dot{T}_{0G,S1} = \begin{bmatrix} -\dot{q}_4\sin(q_4)\cos(q_5) - \dot{q}_5\sin(q_5)\cos(q_4) \\ \dot{q}_3(-\sin(q_3)\sin(q_5) + \sin(q_4)\cos(q_3)\cos(q_5)) + \dot{q}_4\sin(q_3)\cos(q_4)\cos(q_5) + \dot{q}_5(-\sin(q_3)\sin(q_4)\sin \\ \dot{q}_3(\sin(q_3)\sin(q_4)\cos(q_5) + \sin(q_5)\cos(q_3)) - \dot{q}_4\cos(q_3)\cos(q_4)\cos(q_5) + \dot{q}_5(\sin(q_3)\cos(q_5) + \sin \\ 0 \end{bmatrix}$$

$$\dot{T}_{refG,S2/S1} = \begin{bmatrix} -\dot{q}_6 \sin(q_6) & 0 & \dot{q}_6 \cos(q_6) & 0 \\ 0 & 0 & 0 & 0 \\ -\dot{q}_6 \cos(q_6) & 0 & -\dot{q}_6 \sin(q_6) & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\dot{T}_{refG,S3/S1} = \begin{bmatrix} -\dot{q}_7 \sin(q_7) & 0 & \dot{q}_7 \cos(q_7) & 0 \\ 0 & 0 & 0 & 0 \\ -\dot{q}_7 \cos(q_7) & 0 & -\dot{q}_7 \sin(q_7) & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\dot{T}_{refG,S4/S1} = \begin{bmatrix} -25\dot{q}_4 \sin(25q_4 - 25q_6) + 25\dot{q}_6 \sin(25q_4 - 25q_6) & 0 & -25\dot{q}_4 \cos(25q_4 - 25q_6) + 25\dot{q}_6 \cos(25q_4 - \\ 0 & 0 & 0 \\ 25\dot{q}_4 \cos(25q_4 - 25q_6) - 25\dot{q}_6 \cos(25q_4 - 25q_6) & 0 & -25\dot{q}_4 \sin(25q_4 - 25q_6) + 25\dot{q}_6 \sin(25q_4 - \\ 0 & 0 & 0 \end{bmatrix}$$

$$\dot{T}_{refG,S5/S1} = \begin{bmatrix} -25\dot{q}_4 \sin(25q_4 - 25q_7) + 25\dot{q}_7 \sin(25q_4 - 25q_7) & 0 & -25\dot{q}_4 \cos(25q_4 - 25q_7) + 25\dot{q}_7 \cos(25q_4 - \\ 0 & 0 & 0 \\ 25\dot{q}_4 \cos(25q_4 - 25q_7) - 25\dot{q}_7 \cos(25q_4 - 25q_7) & 0 & -25\dot{q}_4 \sin(25q_4 - 25q_7) + 25\dot{q}_7 \sin(25q_4 - \\ 0 & 0 & 0 \end{bmatrix}$$

The velocity of the center of gravity of each body

$$\vec{v}_{G,S0} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\vec{v}_{G,S1} = \begin{bmatrix} \dot{q}_0 + 0.9\dot{q}_4 \cos(q_4) \\ \dot{q}_1 - 0.9\dot{q}_3 \cos(q_3)\cos(q_4) + 0.9\dot{q}_4 \sin(q_3)\sin(q_4) \\ \dot{q}_2 - 0.9\dot{q}_3 \sin(q_3)\cos(q_4) - 0.9\dot{q}_4 \sin(q_4)\cos(q_3) \end{bmatrix}$$

$$\left\{\vec{v}_{G,S2/S1}\right\}_{S1} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\left\{\vec{v}_{G,S3/S1}\right\}_{S1} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\left\{\vec{v}_{G,S4/S1}\right\}_{S1} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\left\{\vec{v}_{G,S5/S1}\right\}_{S1} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

The acceleration of the center of gravity of each body

$$\vec{a}_{G,S0} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\vec{a}_{G,S1} = \begin{bmatrix} -0.9\dot{q}_4^2 \sin(q_4) + \ddot{q}_0 + 0.9\ddot{q}_4 \cos(q_4) \\ 0.9\dot{q}_3^2 \sin(q_3)\cos(q_4) + 1.8\dot{q}_3\dot{q}_4 \sin(q_4)\cos(q_3) + 0.9\dot{q}_4^2 \sin(q_3)\cos(q_4) + \ddot{q}_1 - 0.9\ddot{q}_3 \cos(q_3)\cos(q_4) + 0 \\ -0.9\dot{q}_3^2 \cos(q_3)\cos(q_4) + 1.8\dot{q}_3\dot{q}_4 \sin(q_3)\sin(q_4) - 0.9\dot{q}_4^2 \cos(q_3)\cos(q_4) + \ddot{q}_2 - 0.9\ddot{q}_3 \sin(q_3)\cos(q_4) - \end{bmatrix}$$

$$\left\{\vec{a}_{G,S2/S1}\right\}_{S1} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\left\{\vec{a}_{G,S3/S1}\right\}_{S1} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\left\{\vec{a}_{G,S4/S1}\right\}_{S1} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\left\{\vec{a}_{G,S5/S1}\right\}_{S1} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

The rotation velocity of each body

$$\vec{\omega}_{G,S0} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\vec{\omega}_{G,S1} = \begin{bmatrix} \dot{q}_3 + \dot{q}_5 \sin(q_4) \\ \dot{q}_4 \cos(q_3) - \dot{q}_5 \sin(q_3) \cos(q_4) \\ \dot{q}_4 \sin(q_3) + \dot{q}_5 \cos(q_3) \cos(q_4) \end{bmatrix}$$

$$\left\{\vec{\omega}_{G,S2/S1}\right\}_{S1} = \begin{bmatrix} 0 \\ \dot{q}_6 \\ 0 \end{bmatrix}$$

$$\left\{\vec{\omega}_{G,S3/S1}\right\}_{S1} = \begin{bmatrix} 0 \\ \dot{q}_7 \\ 0 \end{bmatrix}$$

$$\left\{\vec{\omega}_{G,S4/S1}\right\}_{S1} = \begin{bmatrix} 0 \\ -25\dot{q}_4 + 25\dot{q}_6 \\ 0 \end{bmatrix}$$

$$\left\{\vec{\omega}_{G,S5/S1}\right\}_{S1} = \begin{bmatrix} 0 \\ -25\dot{q}_4 + 25\dot{q}_7 \\ 0 \end{bmatrix}$$

The rotation acceleration of each body

$$\dot{\vec{\omega}}_{G,S0} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\dot{\vec{\omega}}_{G,S1} = \begin{bmatrix} \dot{q}_4 \dot{q}_5 \cos(q_4) + \ddot{q}_3 + \ddot{q}_5 \sin(q_4) \\ -\dot{q}_3 \dot{q}_4 \sin(q_3) - \dot{q}_3 \dot{q}_5 \cos(q_3) \cos(q_4) + \dot{q}_4 \dot{q}_5 \sin(q_3) \sin(q_4) + \ddot{q}_4 \cos(q_3) - \ddot{q}_5 \sin(q_3) \cos(q_4) \\ \dot{q}_3 \dot{q}_4 \cos(q_3) - \dot{q}_3 \dot{q}_5 \sin(q_3) \cos(q_4) - \dot{q}_4 \dot{q}_5 \sin(q_4) \cos(q_3) + \ddot{q}_4 \sin(q_3) + \ddot{q}_5 \cos(q_3) \cos(q_4) \end{bmatrix}$$

$$\left\{\dot{\vec{\omega}}_{G,S2/S1}\right\}_{S1} = \begin{bmatrix} 0 \\ \ddot{q}_6 \\ 0 \end{bmatrix}$$

$$\left\{\dot{\vec{\omega}}_{G,S3/S1}\right\}_{S1} = \begin{bmatrix} 0 \\ \ddot{q}_7 \\ 0 \end{bmatrix}$$

$$\left\{\dot{\vec{\omega}}_{G,S4/S1}\right\}_{S1} = \begin{bmatrix} 0 \\ -25\ddot{q}_4 + 25\ddot{q}_6 \\ 0 \end{bmatrix}$$

$$\left\{\dot{\vec{\omega}}_{G,S5/S1}\right\}_{S1} = \begin{bmatrix} 0 \\ -25\ddot{q}_4 + 25\ddot{q}_7 \\ 0 \end{bmatrix}$$

3 Definition of external forces

External forces have been defined in the procedure `AddAppliedEfforts()`.

4 Simulation

The routine `NewmarkIntegration` performs the integration of the equations of motion up to time *FinalTime* by regular time intervals equal to *StepSave* and with the maximum allowed time step *StepMax* defined in the file segway.cpp. The following values are used:

- *FinalTime* is the simulation duration (=15 s),
- *StepSave* is the time step in the numerical integration (=0.01 s),
- *StepMax* is the maximum allowed time step (=1e-05 s),

the initial conditions being all zero.

5 Results

The time evolution of the different configuration parameters and their first and second derivatives can easily be plotted by Gnuplot as seen in figures 1 to 3 with the code listed below:

```
reset
set xlabel "Time [s]"
set grid
```

```
set term postscript eps color "Times-Roman" 20
set output "figure1.eps"
set ylabel "displacements"
plot 'segway.res' using 1:2 title 'q_0' with line , 'segway.res' using 1:5 title 'q_1' with line , 'segway.res'
set term pop
replot
pause -1 'Next plot (velocity level)?'

set term postscript eps color "Times-Roman" 20
set output "figure2.eps"
set ylabel "velocities"
plot 'segway.res' using 1:3 title 'qd_0' with line , 'segway.res' using 1:6 title 'qd_1' with line , 'segway.re
set term pop
replot
pause -1 'Next plot (acceleration level)?'

set term postscript eps color "Times-Roman" 20
set output "figure3.eps"
set ylabel "accelerations"
plot 'segway.res' using 1:4 title 'qdd_0' with line , 'segway.res' using 1:7 title 'qdd_1' with line , 'segway.
set term pop
replot
pause -1
```

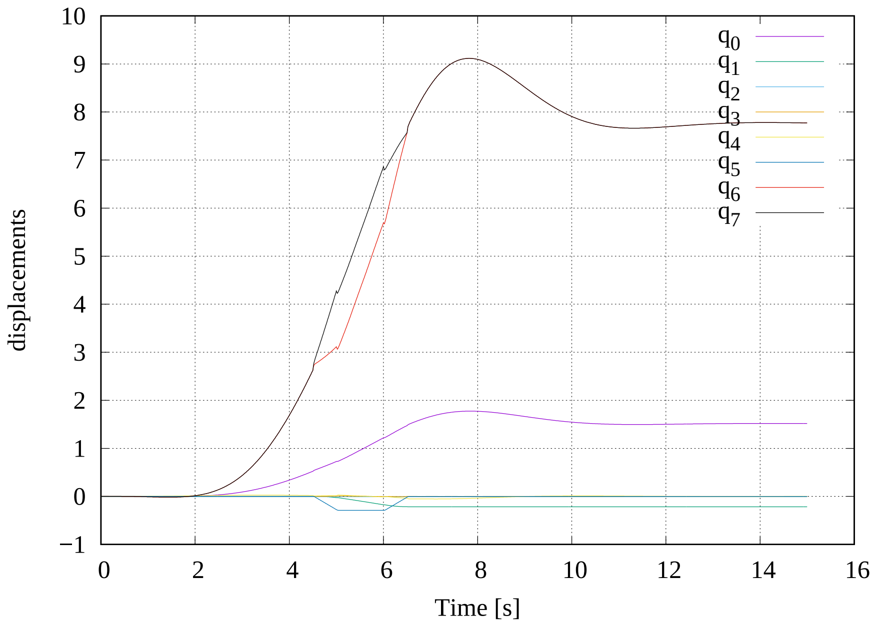


Figure 1: Time evolution of configuration parameters

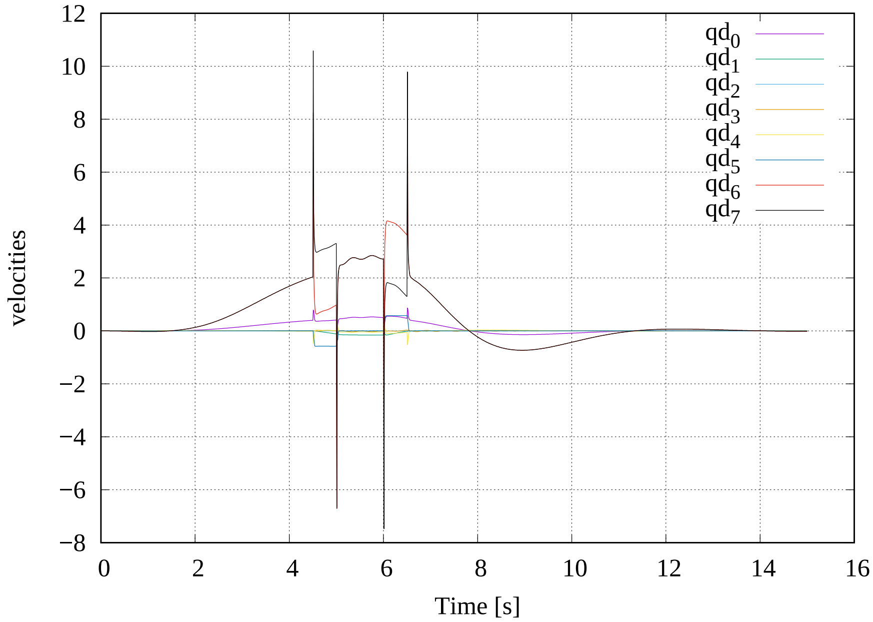


Figure 2: Time evolution of time derivatives of configuration parameters

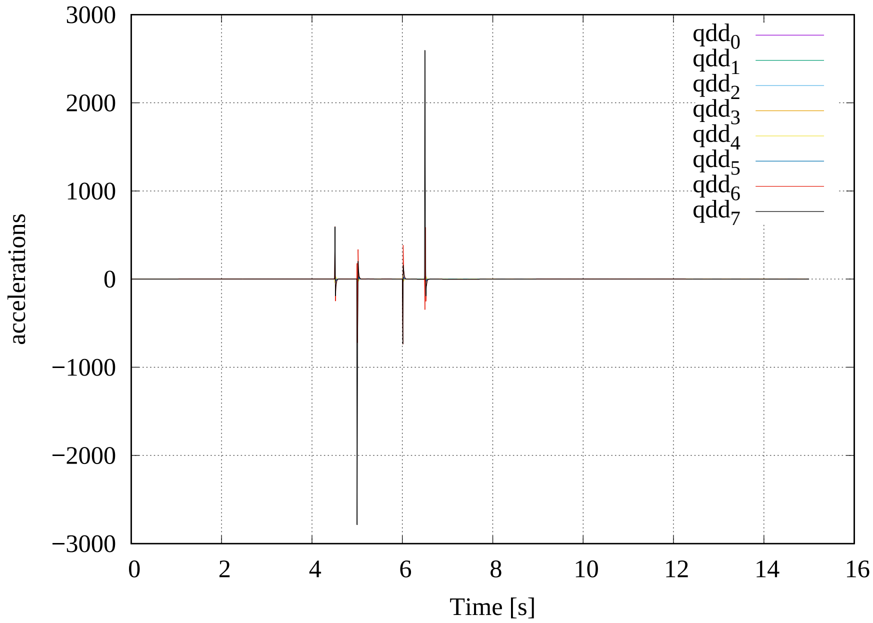


Figure 3: Time evolution of second time derivatives of configuration parameters

A User's Python code

```
#!/usr/bin/python3

import sys
sys.path.insert(0, '../../../..')
sys.dont_write_bytecode = True # Disable pycache folder

from cagem import * # Import functions to derive system kinematics

#~~~~~~~~~~~~~~~~~#
# Segway Regulation Control #
#~~~~~~~~~~~~~~~~~#

#-------------------#
# System properties #
#-------------------#
MBSyst=MBSClass(nbrbody=6,\
nbrdof=8,\
nbrdep=0,\
    ApplicationTitle="Segway Regulation Control",\
    ApplicationFileName="segway")
# nbrbody: Number of bodies
# nbrdof:  Number of degrees of freedom
# nbrdep:  Number of dependent parameters
q,p,pexpr,t=MBSyst.Getqpt() # Declare system variables

# Gravity vector
MBSyst.SetGravity(0,0,-9.81)

# Write in cpp format
# Set global variables
MBSyst.SetGlobalVar("""
/// Geometrical constants
double L=0.9;
double r=0.2;
double c=0.4;
double a=0.25;
double n=25.0;

/// Electrical constants
double Km =0.085;
double w_left, w_right;
double i_left, i_right;
double R = 0.35;

/// Control
double desired_wz;
double actual_wz;
double error_wz;

double u_steer;
double u_left;
double u_right;

double K_wz = 30.0;

double SetPoint;

/// Time parameters
double t_total = 15.0; // s
double dt=0.01; // s
```

```
/// Forward control constants
double CurrentPosition_Z;
double CurrentVelocity_Z;
double CurrentAcceleration_Z;
double Z_Axis_init = 0; // [m]

double TCP_Angular_velocity = 3.5; // [rad/s]
double Duration_Accel = 2.0; // [s] We want the desired velocity after half of a revolution
double TCP_velocity_Z=0.5;
double PostPosition_Z=0.0;
double Duration_Decel = M_PI/(0.5*TCP_Angular_velocity);
double Duration_CstVelocity = 3.0; // For the constant velocity
double Duration_Rest = 9.0;


///************************************///
///  Segment: Linear velocity profile  ///
///************************************///

//--------------------------------------------------

void TrajLinVel_Segment(bool Reverse,double &dx, double desired_vel_TCP, double t_accel, double t,
                        double &xd_end, double &xdd_end, double t_init, double x_init, double &x_final)
{
    // Compute position profile so that the velocity has trapezoidal shape
    // No blend between segments

    // dx              : computed position increment to be added to the initial position
    // desired_vel_TCP : desired TCP velocity in m/s
    // t_accel         : time duration during which acceleration is constant
    // t               : current time of EasyDdyn simulation
    // xd_end          : computed TCP velocity (theoretical)
    // xdd_end         : computed TCP acceleration (theoretical)
    // t_init          : initial time when calling this function for the first time
    // x_init          : initial position of the end effector

    // Time parameter
    double t_linear_acc = t_accel;

    // End effector velocity
    double FeedRate_SI = desired_vel_TCP;

    // Initial acceleration
    double a = desired_vel_TCP/t_accel;

    // Current variables to compute the kinematic profile
    double Acc_t;
    double Vel_t;
    double Position_t;
    double t_phase;

    // Precomputation of the constants (for linear velocity and linear deceleration)
    // constants for linear velocity
    double a0_phase1;
    if(Reverse == false)
    {
        a0_phase1=a;
    }
    else
    {
        a0_phase1=-a;
    }
```

```
    //double a0_phase1=a;

    double v0_phase1=0.0;
    double x0_phase1= x_init;

    // Computation of increments
    if( ((t-t_init)<t_accel) && ((t-t_init)>=0))
    {
        // phase 1: constant acceleration
        if( (t-t_init)<=t_linear_acc)
        {
            t_phase = t-t_init;
            Acc_t = a0_phase1;
            Vel_t = v0_phase1 + a0_phase1*t_phase;
            Position_t = x0_phase1 + v0_phase1*t_phase + (1.0/2.0)*a0_phase1*pow(t_phase,2);
        }
    }
    else if( (t-t_init)<0.0 )
    {
        Acc_t = xdd_end;
        Vel_t= xd_end;
        Position_t = dx;
    }
    else
    {
        Acc_t = 0.0;
        Vel_t= 0.0;
        Position_t = 0.0;
    }

    // Final position after duration of constant jerk
    x_final = x0_phase1 + v0_phase1*t_accel + (1.0/2.0)*a0_phase1*pow(t_accel,2);

    // save current acceleration velocity and position
    xdd_end = Acc_t;
    xd_end = Vel_t;
    dx = Position_t;
}

///**********************************///
///  Segment: Constant velocity profile  ///
///**********************************///

//---------------------------------------------------

void TrajConstVel_Segment(double &dx, double desired_vel_TCP, double t_CstVel, double t,
                        double &xd_end, double &xdd_end, double t_init, double x_init, double &x_final)
{
    // Constant velocity over a particular duration t_CstVel

    // dx              : computed position increment of the TCP to be added to the initial position of the TCP
    // desired_vel_TCP : desired TCP velocity in m/s
    // t_CstVel         : time duration during which velocity of TCP is constant
    // t               : current time of EasyDdyn simulation
    // xd_end          : computed TCP velocity (theoretical)
    // xdd_end         : computed TCP acceleration (theoretical)
    // t_init          : initial time when calling this function for the first time
    // x_init          : initial position of the end effector

    // endtraj: last segment for rest, keep same values

    // End effector velocity
```

```
    double FeedRate_SI = desired_vel_TCP;

    // Current variables to compute the kinematic profile
    // double Jerk_t
    double Acc_t;
    double Vel_t;
    double Position_t;
    double t_phase;

    // phase 1: constant velocity
    if( ((t-t_init)<t_CstVel) && ((t-t_init)>=0)) // Constant velocity
    {

        t_phase = t-t_init;
        Acc_t = 0.0;
        Vel_t = FeedRate_SI;
        Position_t = x_init + FeedRate_SI*t_phase;

    }
    else if( (t-t_init)<0.0 )
    {
        Acc_t = xdd_end;
        Vel_t= xd_end;
        Position_t = dx;
    }
    else
    {
        Acc_t = 0.0;
        Vel_t= 0.0;
        Position_t = 0.0;
    }

    // Final position after duration of constant jerk
    x_final = x_init + FeedRate_SI*t_CstVel;

    // save current acceleration velocity and position
    xdd_end = Acc_t;
    xd_end = Vel_t;
    dx = Position_t;
}

//---------------------------------------------------
""")

# Some constants
L=0.9
r=0.2
c=0.4
a=0.25
n=25

#--------------------#
# Inertia properties #
#--------------------#
MBSyst.body[0].Set(mass=1e-10,Ixx=1e-10, Iyy=1e-10, Izz=1e-10)
MBSyst.body[1].Set(mass=90,  Ixx=21.6,     Iyy=21.6,     Izz=2.5)
MBSyst.body[2].Set(mass=15,  Ixx=0.2,     Iyy=0.3,     Izz=0.2)
MBSyst.body[3].Set(mass=15,  Ixx=0.2,     Iyy=0.3,     Izz=0.2)
MBSyst.body[4].Set(mass=1,  Ixx=1.2e-4,     Iyy=1.0e-4,     Izz=1.2e-4)
MBSyst.body[5].Set(mass=1,  Ixx=1.2e-4,     Iyy=1.0e-4,     Izz=1.2e-4)

#-------------------#
```

```
# Position matrices #
#-------------------#
MBSyst.body[0].T0F=Tdisp(0,0,0)
# passenger
MBSyst.body[1].T0F=Tdisp(q[0],q[1],q[2]+r)*Trotx(q[3])*Troty(q[4])*Trotz(q[5])*Tdisp(0,0,L)
# Left Wheel
MBSyst.body[2].TrefF=Tdisp(0,-c,-L)*Troty(q[6])
MBSyst.body[2].ReferenceFrame(1)
# Right Wheel
MBSyst.body[3].TrefF=Tdisp(0,c,-L)*Troty(q[7])
MBSyst.body[3].ReferenceFrame(1)
# Left rotor
MBSyst.body[4].TrefF=Tdisp(0,-a,-L)*Troty(n*(q[6]-q[4]))
MBSyst.body[4].ReferenceFrame(1)
# Right rotor
MBSyst.body[5].TrefF=Tdisp(0,a,-L)*Troty(n*(q[7]-q[4]))
MBSyst.body[5].ReferenceFrame(1)

#-----------------------#
# Initial configuration #
#-----------------------#

#---------------------------#
# Forces #
#---------------------------#
MBSyst.Force("""

/// Steering command

/// Desired steering
if( t> 4.5 && t<5.0) {
    desired_wz = 3.0; /// rad/s
} else if( t> 6.0 && t<6.5) {
    desired_wz = -3.0; /// rad/s
} else {
    desired_wz = 0.0;
}

/// Error on angular velocity

actual_wz = (a/r) * (qd[7]-qd[6]);
error_wz = desired_wz - actual_wz;

/// Input for each motor
u_steer= K_wz*error_wz;

u_left = u[0] - u_steer;
u_right = u[0] + u_steer;

/// Saturation
if(u_left>72.0) u[0] = 72.0;
if(u_left<-72.0) u[0] = -72.0;

if(u_right>72.0) u[0] = 72.0;
if(u_right<-72.0) u[0] = -72.0;

/// Motors

w_left=n*(qd[6]-qd[4]);
w_right=n*(qd[7]-qd[4]);

i_left = (u_left-Km*w_left)/R;
```

```
i_right = (u_right-Km*w_right)/R;

body[4].MG += (Km*i_left)*body[4].TOG.R.uy(); // Torque on Rotor 1
body[1].MG -= (Km*i_left)*body[4].TOG.R.uy();

body[5].MG += (Km*i_right)*body[5].TOG.R.uy(); // Torque on Rotor 2
body[1].MG -= (Km*i_right)*body[5].TOG.R.uy();

/// Tyre
structtyre tyre_segway;
tyre_segway.r1=0.2;
tyre_segway.r2=0.03;
tyre_segway.Kz=100000;
tyre_segway.Cz=600;
tyre_segway.Fznom=700;
tyre_segway.Clongnom=10000;
tyre_segway.nlong=0.1;
tyre_segway.Clatnom=10000;
tyre_segway.nlat=0.1;
tyre_segway.Ccambernom=1000;
tyre_segway.ncamber=0.1;
tyre_segway.fClbs=1.0;
tyre_segway.fClbd=0.8;

// Left Wheel
AddTyreEfforts(2,vcoord(0,1,0),tyre_segway);
// Right Wheel
AddTyreEfforts(3,vcoord(0,1,0),tyre_segway);

""")

#------------------------#
# Integration parameters #
#------------------------#
MBSyst.SetIntegrationParameters(tfinal=15, hsave=0.01, hmax=1e-5)
# tfinal = duration of simulation [s]
# hsave = time step for saving results [s]
# hmax = adaptive integration time step [s]

#---------#
# Options #
#---------#
STATIC=1 # Set STATIC to 1 to search the static equilibrium before integration
POLE=1 # Set POLE to 1 to perform an eigen value analysis
TEST=0 # Set TEST to 1 to perform the efficiency tests

MBSyst.ComputeKinematics() # Derive the system kinematics symbolically
MBSyst.EasyDynFlags(STATIC,POLE,TEST) # Define flags
MBSyst.ExportEasyDynProgram() # Write .cpp file enclosing the kinematics

MBSyst.ExportUK_Latex_Report() # Uncomment to generate the English LaTeX report
MBSyst.ExportGnuplotScript() # Uncomment to output script for graphs of variables
```