UR-5 Robot Manipulator

Master Degree in Automation Engineering

Course: Modeling and Simulation of Mechatronic Systems – 2023/2024

Professor:

Alessandro Macchelli

Students:

Andrea Perna

Giuseppe Speciale

Ammar Garooge

Meisam Tavakoli

Introduction

REQUESTED FEATURES

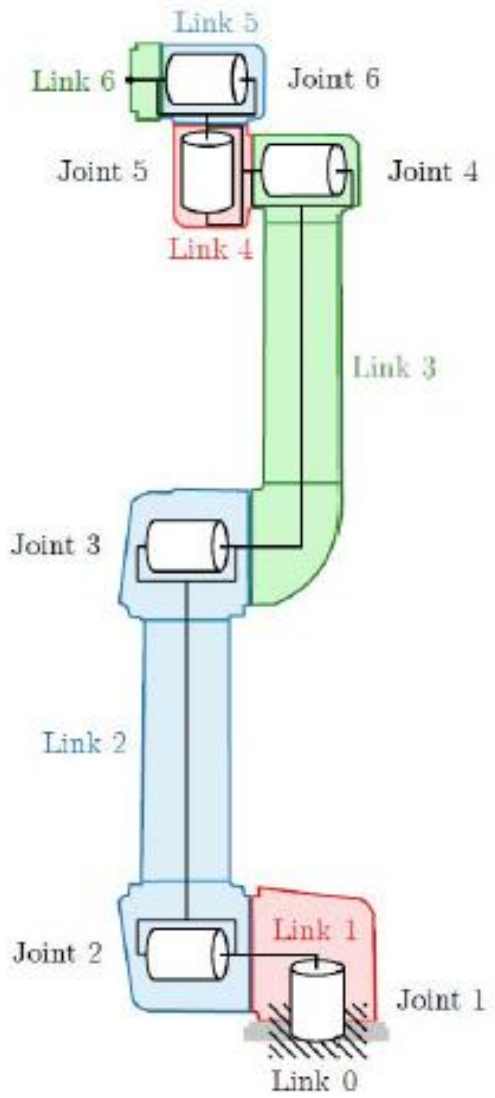
DEVELOPED IN 20SIM

MODULAR STRUCTURE

RETICULATION TECHNIQUE

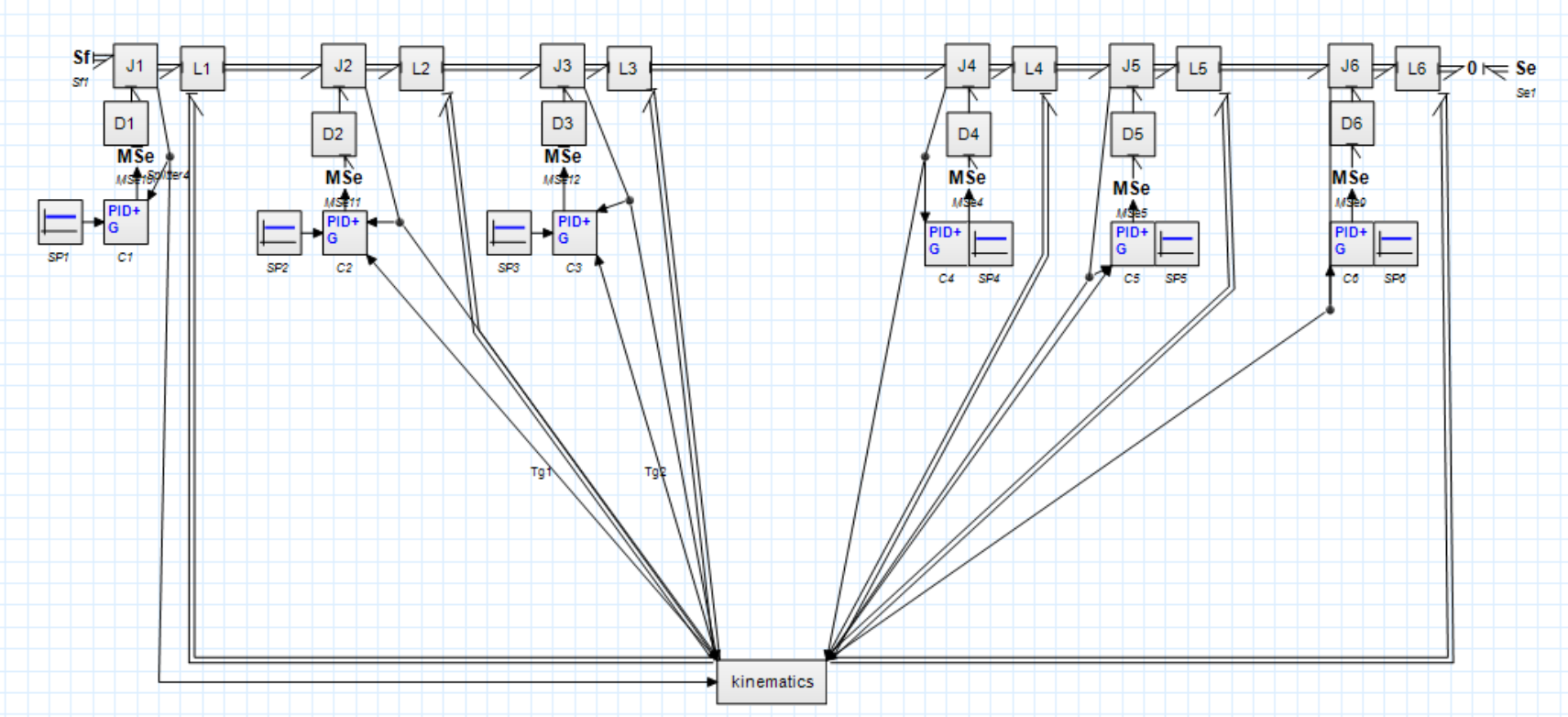
Model Overview

- Modelling and simulation of a **UR5 Manipulator** of Universal Robots;
- **Six revolute joints**: base, shoulder, elbow and wrist (x3), each of them equipped with its own motor, i.e., **no transmission organs** are present;
- **DH parameters** simplifies the derivation of transformation matrices;
- Complete **kinematics**, and **dynamics** of the manipulator;

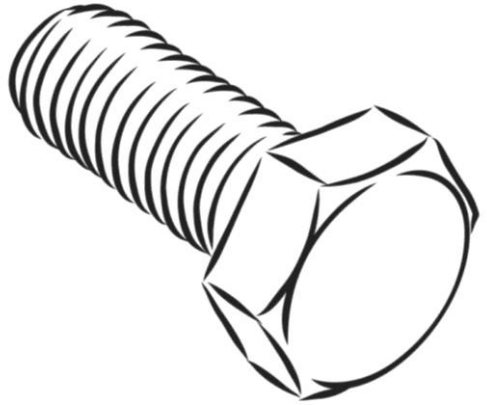


Model in 20Sim

- **Centralized Kinematics** block contains HTM and DH parameters;
- **Modular** and **scalable** structure, with similar link-joint pairs;
- **Setpoints** provide desired configuration reached via **PID control**;
- **Harmonic drive** of the motor connected to each joint;



Twist and Wrenches



In modeling the UR5 robot manipulator, *Screw Theory* has been used, leading to a better understanding of the UR5 manipulator. A six-dimensional representation of variables has been used:

- **Twists** represent the velocity of a rigid body's motion, comprising both linear and angular components;
- **Wrenches** denote the forces and torques acting on a rigid body, also in six dimensions;

$$\begin{pmatrix} v'_p \\ w' \end{pmatrix} = \begin{pmatrix} R_{OG}^T & -r_{GP} \times R_{OG}^T \\ \mathbf{0} & R_{OG}^T \end{pmatrix} \begin{pmatrix} v'_G \\ w' \end{pmatrix}$$

Body frame configuration tensor

Coordinates Transformation

$$H_i^{i-1} = \mathrm{Trans}(z_{i-1}, d_i) \cdot \mathrm{Rot}(z_{i-1}, \theta_i) \cdot \mathrm{Trans}(x_i, a_i) \cdot \mathrm{Rot}(x_i, \alpha_i)$$

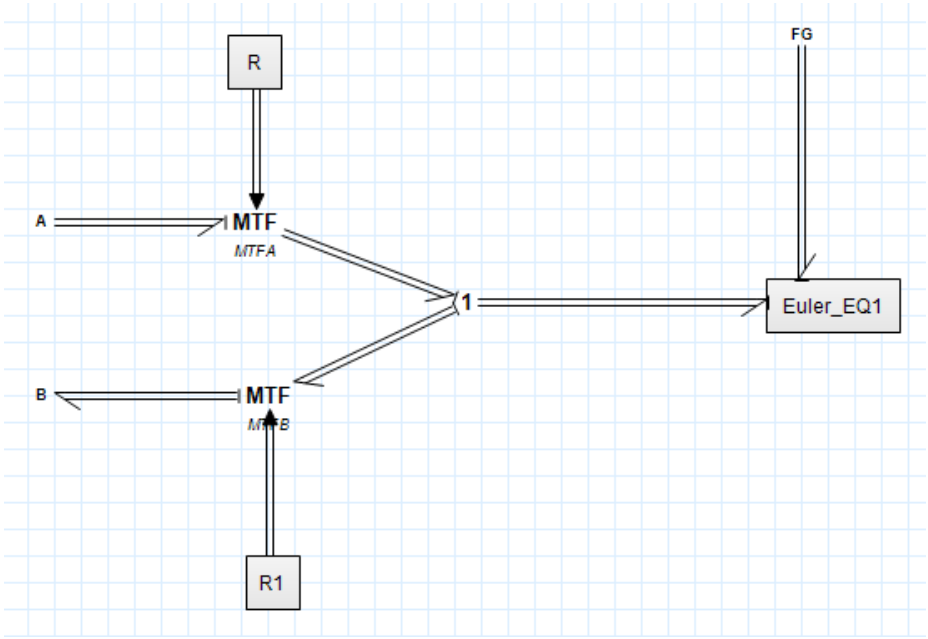
$$= \begin{pmatrix} \cos\theta_i & -\sin\theta_i \cos\alpha_i & \sin\theta_i \sin\alpha_i & a_i \cos\theta_i \\ \sin\theta_i & \cos\theta_i \cos\alpha_i & -\cos\theta_i \sin\alpha_i & a_i \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- To transform coordinates between the two link's ends, some **subsequent transformations** are needed;
- Overall **Homogenous Transformation matrix** is built inside Kinematics;
- The **Adjoint matrix** of the HTM is computed to express the change of coordinates between frames;

Link Model

```
HA_G = [1,0,0,a;
        0,cos(alpha),-sin(alpha),0;
        0,sin(alpha),cos(alpha),d;
        0,0,0,1];
RA_G = HA_G[1:3,1:3];
SP   = skew(rAG)*RA_G;
T_alphaG =       [RA_G[1,1],RA_G[1,2],RA_G[1,3],SP[1,1],SP[1,2],SP[1,3];
                  RA_G[2,1],RA_G[2,2],RA_G[2,3],SP[2,1],SP[2,2],SP[2,3];
                  RA_G[3,1],RA_G[3,2],RA_G[3,3],SP[3,1],SP[3,2],SP[3,3];
                  0,0,0,RA_G[1,1],RA_G[1,2],RA_G[1,3];
                  0,0,0,RA_G[2,1],RA_G[2,2],RA_G[2,3];
                  0,0,0,RA G[3,1],RA G[3,2],RA G[3,3]];
```

- It models the **transformation of energy variables** through the whole rigid body;
- It transforms wrenches from the previous joint to G through the **Adjoint Matrix** of H_{AG} (MTF);
- **Euler Equations** of motion are expressed in G, contributing to the overall rigid body dynamics;
- **Inverted causality** in 1J, as flow is transmitted from G to end-point B of the link (translation);



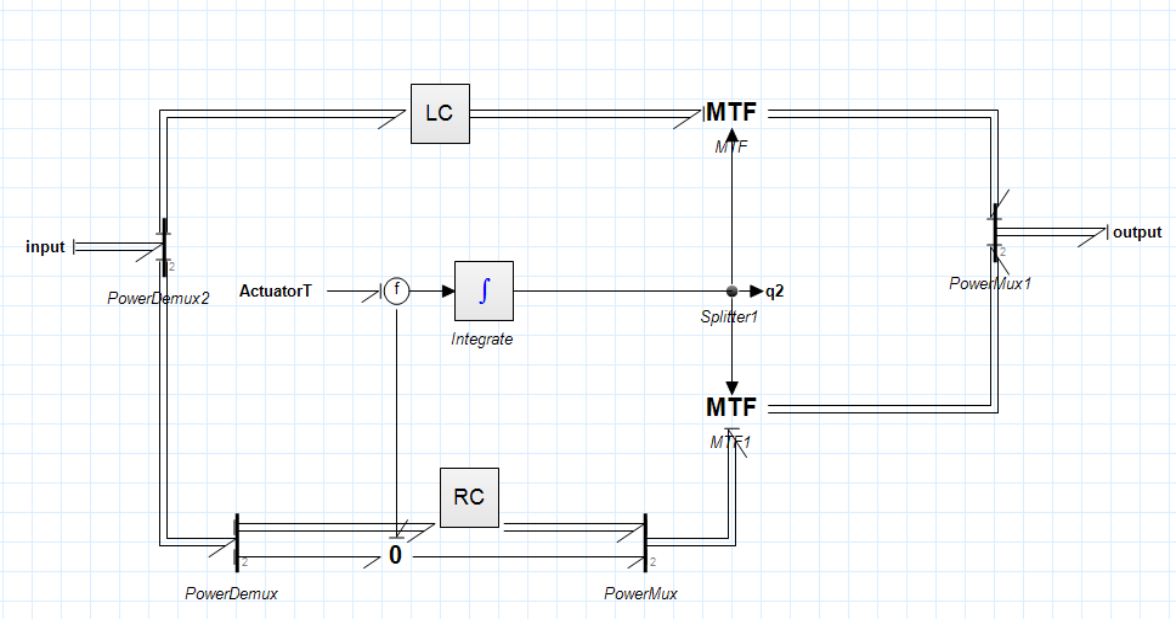
Euler Equations

- They describe the **rotational** and **translational dynamics** of the link, i.e., the rigid body's motion;
- They derive from the **Newton's second laws**;
- They take **applied torques** as inputs to output **angular velocities** and **accelerations**, which set the flow in the 1J.
- Twists and wrenches must be all specified in the **body frame** to be used in the Euler Equations;
- **Gravity** must be included as they exerts a torque on the rigid body, expressed in the centre of gravity G.

```
variables
   // need to take out both velocities
   // angular and linear
   real omega[3,1], v[3,1];
   // write dynamic equations
   real p[3,1],h[3,1];
equations
   v = twist.f[1:3,1];
   omega = twist.f[4:6,1];

   // Euler equations
   p = int(twist.e[1:3,1]+FG.e[1:3,1]-skew(omega)*p);
   h = int(twist.e[4:6,1]-skew(omega)*h);

   R_OG = int(R_OG*skew(omega),R_OG_0);
   P_OG = int(R_OG*v,P_OG_0);
   twist.f[1:3,1] = p/m;
   twist.f[4:6] = inverse(J)*h;
   FG.f[1:3,1]  = twist.f[1:3,1];
   FG.f[4,1]    = 1;
```

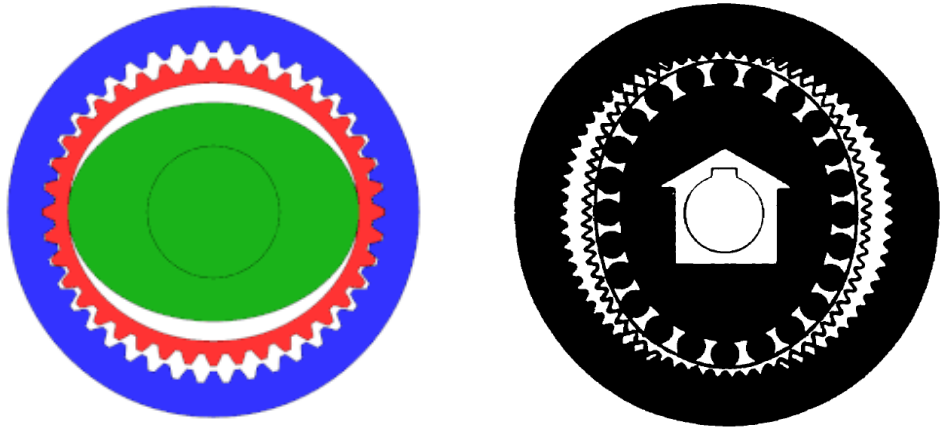
Joint Model

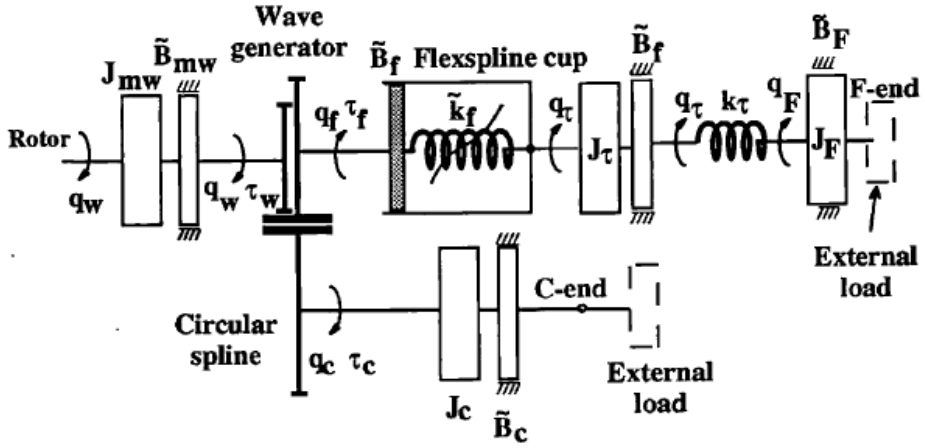
- It represents the **massless point** at the end connecting two consecutive links;
- It takes the wrench as an **input** from the **HD**;
- **No dynamics**: It performs a pure rotation about Z axis;
- It comprises both **linear LC** and **rotational RC** constraints in velocity, by using *"constraint"* 20Sim's function;



Harmonic Drive

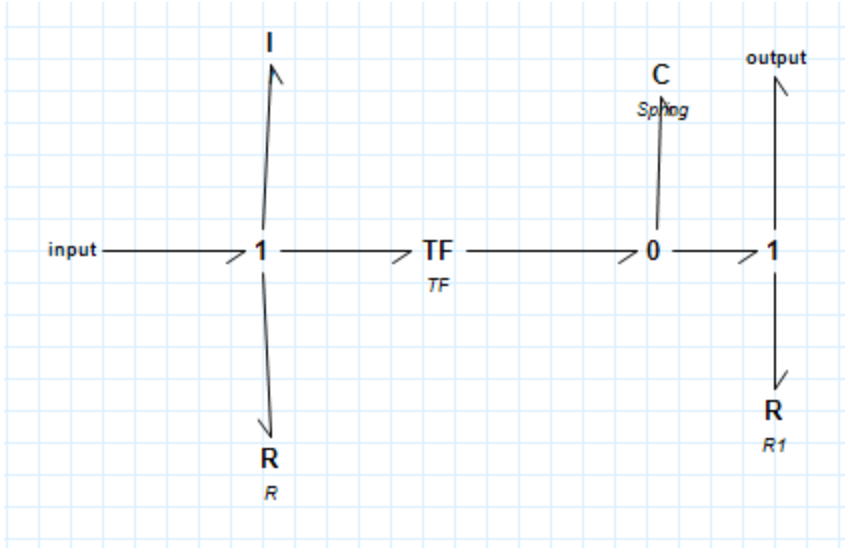
- It is a **mechanical gear device** with a flexible spline and a deformable elliptical plug;
- It works by engaging the inner teeth with the rotating plug, leading to **angular velocity reduction** and **torque increase** for the motor;
- Three components: **non-linear spring**, **two bearings** (dissipation) and **inertia**;

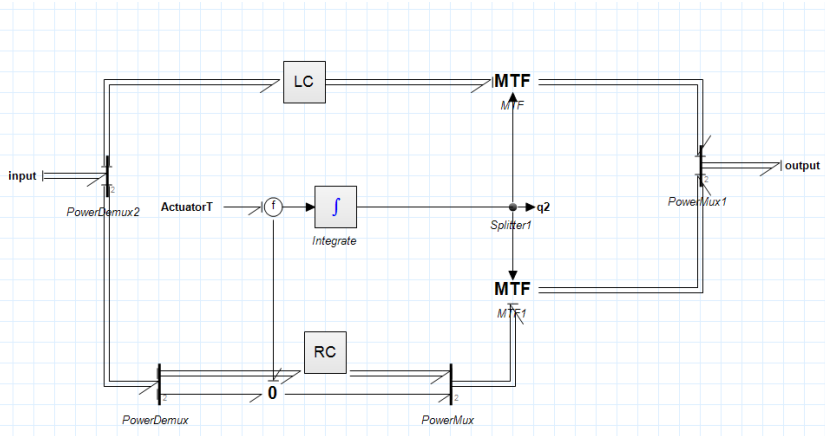




HD: Use in the model

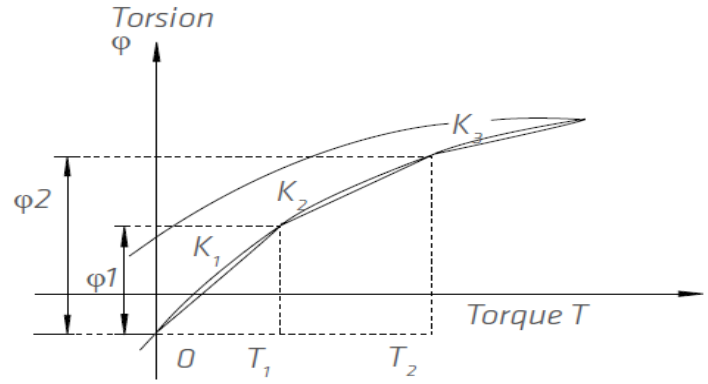
- Harmonic drives are used **to actuate the joints**, providing the necessary torque and precision for control.
- The **compact form** helps in integrating them into the limited space available within the robotic joints;
- Flow integration is performed in joint to get the actual **angles** to use inside the **rotation matrix**;
- Input is a 1D modulated **source of effort** coming from PID controller, output is the torque;





HD: Spring

- **Non-linear C element**, different regions of operation depending on the torque and the torsion angle;
- The **stiffness** changes at specific points, creating piecewise linear segments on torque-torsion graph.
- **Torque** is computed based on **spring's deformation; torsion angle** is computed via **flow integration**;
- **Conditional programming** is used to discriminate the state over the three regions of operation and to apply the appropriate stiffness equation;

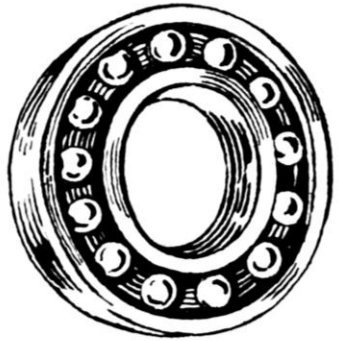


K_1, K_2, K_3 = Torsional stiffness, w = Output angle
$\varphi 1$ = Torsion angle, with output torque T_1
$\varphi 2$ = Torsion angle, with output torque T_2

```
state = int(p.f);

phi1 = T1/k1;
phi2 = T1/k1+ (T2-T1)/k2;

if state <= phi1 or state >= -phi1 then   // first region
   p.e = state * k1;

elsif state > phi2 or state < -phi2 then // third region
   p.e = (state -sign(state)*phi2)*k3 +sign(state)*T2;

else                                          // second region
   p.e = (state -sign(state)*phi1)*k2 +sign(state)*T1;
end;
```

HD: Bearings

- Modelled as **R elements** representing resistive forces due to friction and damping.
- Two different **damping coefficients** are used: B+ for positive velocity and B− for negative velocity, B is the nominal coefficient;
- Each is a **non-linear hysteresis-type operator** that depends on microscopic surface-contact forces;
- Both two are modelled upon **viscous friction** and **Coulomb friction** and have the same structure;



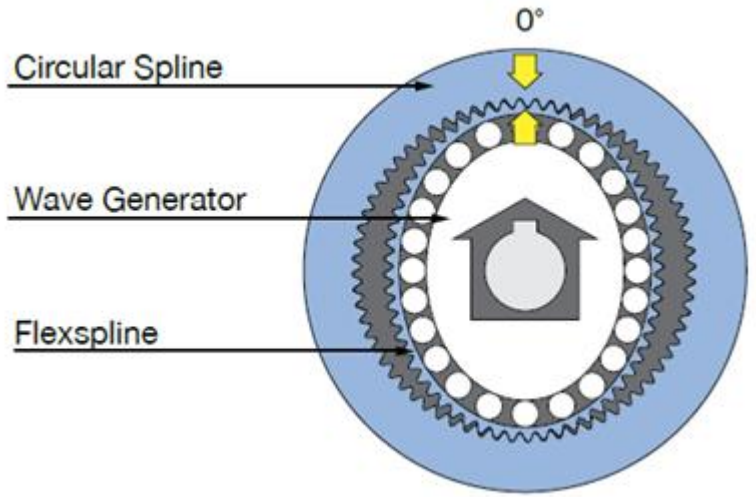
$$\tilde{B}(\dot{q}) = B\dot{q} + (B^+, B^-)sgn(\dot{q})$$

```
if flow > 0 then
   p.e = B*p.f + B_plus;
else
   p.e = B*p.f + B_minus;
end;
```

HD: Transformer

- It is used to model the **gear ratio**, which is crucial to represent the HD's advantage;
- The value **r** is the **negative ratio** of the output to the input speed as well as the ratio of the input to the output torque;
- It ensures **conservation of power,** constant product of torque and angular speed;
- It provides **higher torque** at **lower speed** on the output side;

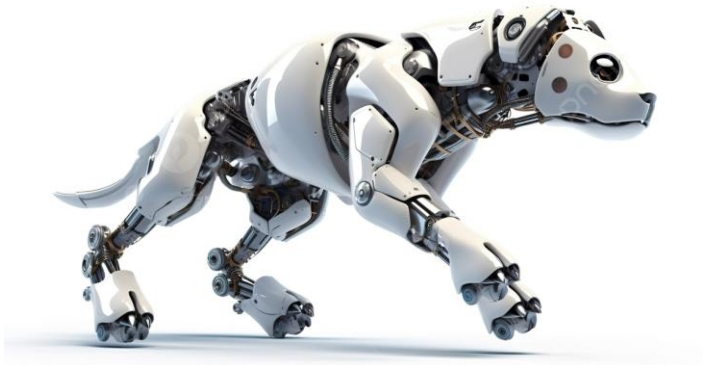
```
parameters
    real r = -100;
equations
    p2.e = r * p1.e;
    p1.f = r * p2.f;
```



HD: Inertia

- Affects **angular acceleration** and system stability;
- Stores **rotational kinetic energy**, helping maintain motion during power fluctuations;
- It is positioned before the wave generator as it affects the initial input of the system's dynamics.
- It is crucial for **optimal performance** and **efficiency**;
- Lower inertia improves response time and helps minimize vibrations for smoother operation.

```
parameters
    real i = 0.091e-4;
    //kg*m^2
equations
     state = int(p.e);
     p.f = state / i;
```



Gravity

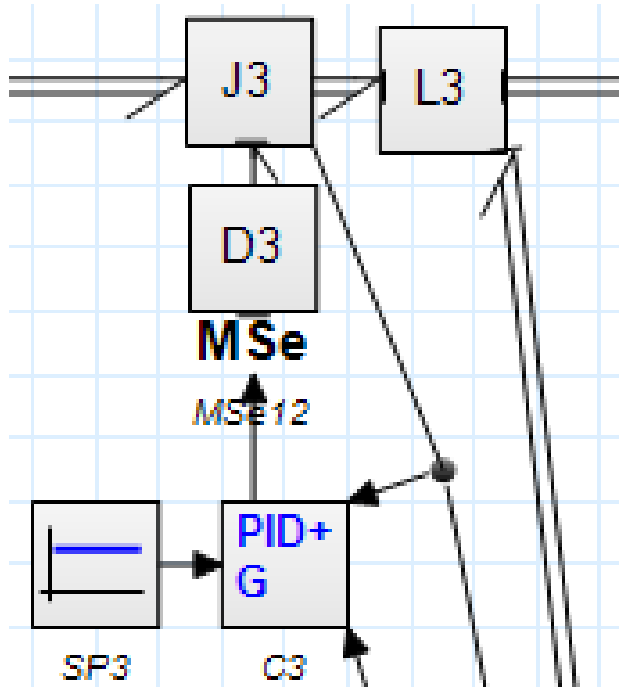
- It is initially defined in the **inertial frame**, as it has a constant direction regardless body's orientation;
- It is then transformed to **body frame** using HTM of each link, with rotation only;
- Transforming gravity to the body frame ensures that the gravitational force is correctly **aligned** with the **body's orientation**, crucial for Euler Equations;
- **Gravity compensation** involves calculating torques required at each joint to counteract the effect of gravity, by improving **stability** and **precision** of **control**;

```
// gravity forces in the inertial frame
real fg1[4,1] = [0;0;-m1*g;0];
real fg2[4,1] = [0;0;-m2*g;0];
real fg3[4,1] = [0;0;-m3*g;0];
real fg4[4,1] = [0;0;-m4*g;0];
real fg5[4,1] = [0;0;-m5*g;0];
real fg6[4,1] = [0;0;-m6*g;0];

//gravity forces in body frame
GravityYes = 1;
FG1.e =transpose(H1_0)*fg1*GravityYes;
FG2.e =transpose(H2_0)*fg2*GravityYes;
FG3.e =transpose(H3_0)*fg3*GravityYes;
FG4.e =transpose(H4_0)*fg4*GravityYes;
FG5.e =transpose(H5_0)*fg5*GravityYes;
FG6.e =transpose(H6_0)*fg6*GravityYes;
flowg1=FG1.f;
flowg2=FG2.f;
flowg3=FG3.f;
flowg4=FG4.f;
flowg5=FG5.f;
flowg6=FG6.f;
```

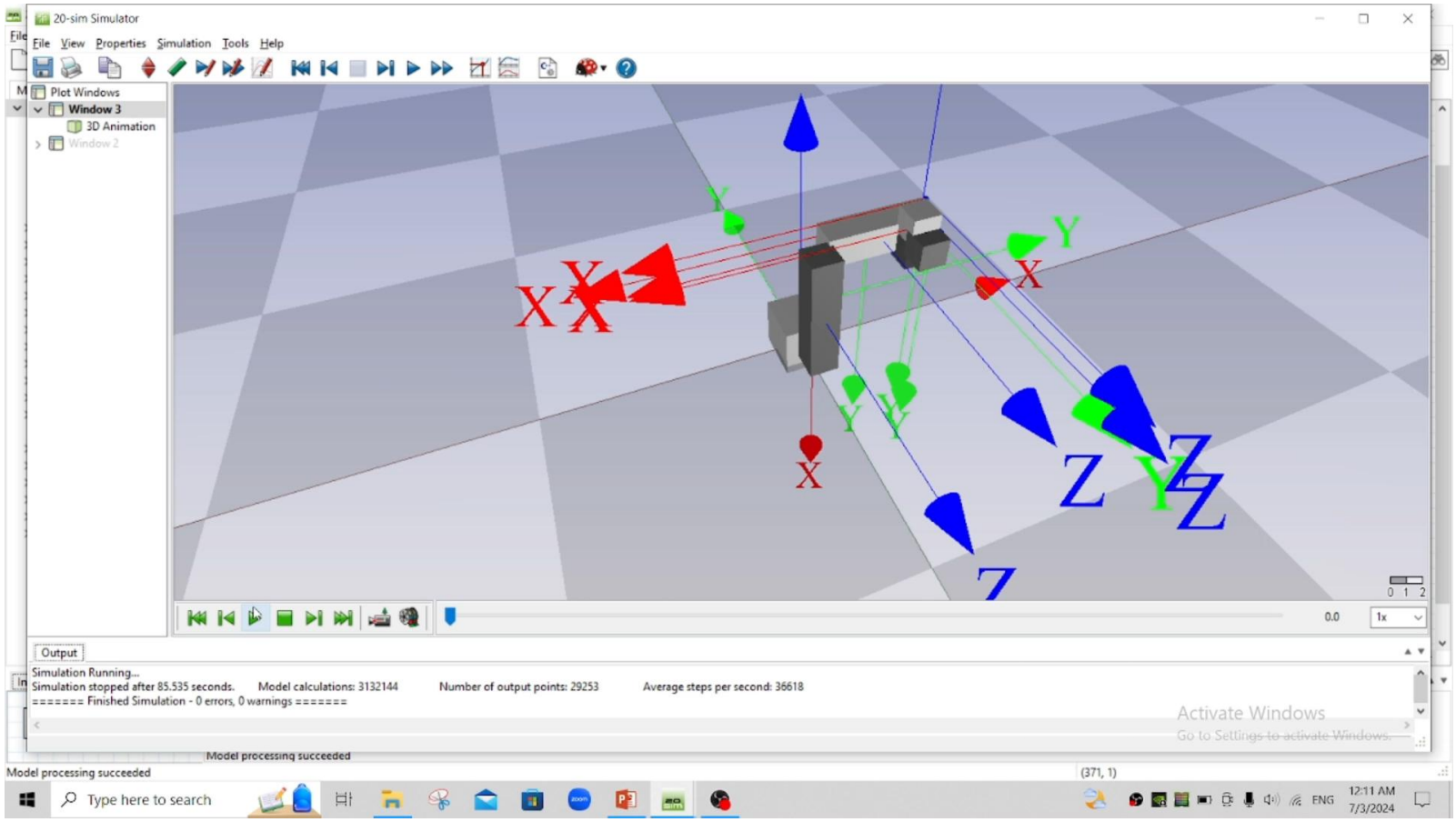
PID+G Control

- PID controller are used for each joint to **control** the **joint angles** to the **desired set points** in a robust way;
- The PID's output acts as **input for the HD**, which then actuates the joint;
- It inherently **learns** and **compensates** for **gravity** by adjusting its output to counteract the gravitational forces acting on the associated joint;
- The **output limits** (minimum and maximum) are set to prevent the controller from commanding a possibly damaging or saturating input for the actuator;
- It continuously adjusts its output in **real-time** based on the **feedback** from the joint angle sensor, maintaining precise control under varying conditions.

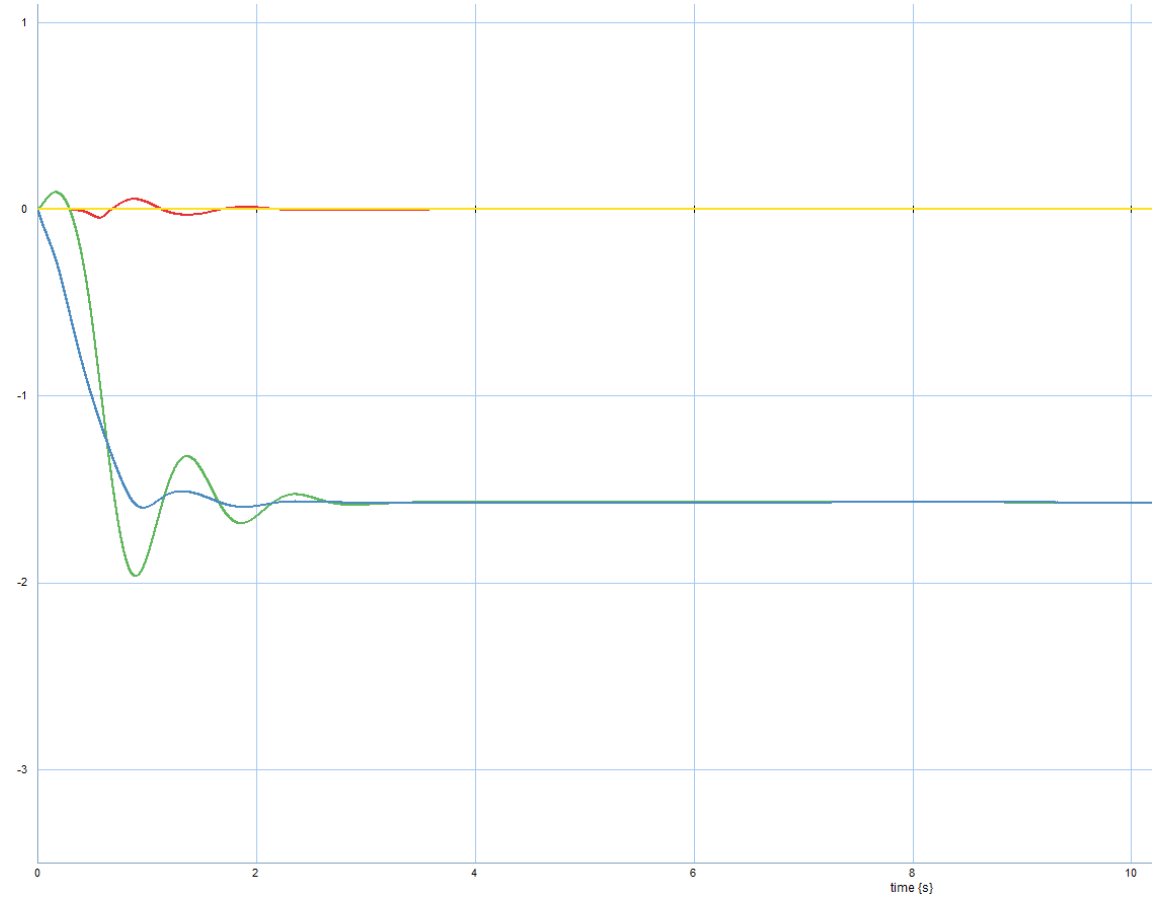
```
parameters
  real K = -2.0 {};      // Proportional gain (increased)
  real Td = 0.05 {s};    // Derivative time constant
  real N = 20 {};        // Derivative gain limitation
  real Ti = 0.5 {s};     // Integral time constant
  real b = 0.2 {};       // Proportional setpoint weighting parameter
  real c = 0.2 {};       // Derivative setpoint weighting parameter
  real Ta = 0.1 {s};     // Tracking time constant
  real minimum = -1 {}; // Minimum controller output
  real maximum = 1 {};// Maximum controller output
variables
  real error,PB_high,PB_low;
  real hidden uP,uI,uD,uDstate,ideal_output;
equations
  error = SP - MV;
  uP = K * ( b * SP - MV );
  uI = int ( K * error / Ti  -  ( ideal_output - output ) / Ta );
  uDstate = int ( uD * N / Td );
  uD = K * (c * SP - MV ) * N - uDstate;
  ideal_output = uP + uI + uD+gravity_compensation/100;
  output = limit (ideal_output, minimum, maximum);
  PB_low = b * SP + (uI + uD - maximum)/K;
  PB_high = b * SP + (uI + uD - minimum)/K;
```



Simulation


20-sim Simulator
File View Properties Simulation Tools Help
Plot Windows
Window 3
3D Animation
Window 2
Output
Simulation Running...
Simulation stopped after 85.535 seconds. Model calculations: 3132144 Number of output points: 29253 Average steps per second: 36618
======= Finished Simulation - 0 errors, 0 warnings =======
Model processing succeeded
Activate Windows
Go to Settings to activate Windows.
Model processing succeeded
(371, 1)
Type here to search
ENG
12:11 AM
7/3/2024

Angles Plots



Thank you for the attention!