Non-Linear Spacecraft Attitude Dynamics and Control Modelling

Michal Jagodzinski

Test document, source: Non-Linear Spacecraft Attitude Dynamics and Control Modelling

Defining the Non-Linear Model

```
using CairoMakie, AlgebraOfGraphics
using ModelingToolkit, ModelingToolkitStandardLibrary
using DifferentialEquations
set_aog_theme!()

@parameters t
const Rot = ModelingToolkitStandardLibrary.Mechanical.Rotational
const B = ModelingToolkitStandardLibrary.Blocks
```

```
@component function SpacecraftAttitude(
    ; name, Jx=100.0, Jy=100.0, Jz=100.0, u0=zeros(3),  0=zeros(3), ˙0=zeros(3)
)

    @named Mx = B.RealInput()
    @named My = B.RealInput()
    @named Mz = B.RealInput()

    @named phi_x = B.RealOutput()
    @named phi_y = B.RealOutput()
    @named phi_z = B.RealOutput()

    sts = @variables  (t)=u0[1]  (t)=u0[2]  (t)=u0[3]  x(t)= 0[1]  y(t)= 0[2]  z(t)= 0[3] ˙x(t)
```

```
    ps = @parameters Jx=Jx Jy=Jy Jz=Jz u0=u0  0= 0 ˙0=˙0

    D = Differential(t)

    eqs = [
        phi_x.u ~  ,
        phi_y.u ~  ,
        phi_z.u ~  ,

        D( ) ~  x +  z * tan( )*cos( ) +  y*tan( )*sin( ),
        D( ) ~  y*cos( ) -  z*sin( ),
        D( ) ~  z*sec( )*cos( ) +  y*sec( )*sin( ),

        D( x) ~ ˙x,
        D( y) ~ ˙y,
        D( z) ~ ˙z,

        Jx * ˙x ~ Mx.u + (Jy - Jz)* y* z,
        Jy * ˙y ~ My.u + (Jz - Jx)* x* z,
        Jz * ˙z ~ Mz.u + (Jx - Jy)* x* y,
    ]

    compose(
        ODESystem(eqs, t, sts, ps; name = name), Mx, My, Mz, phi_x, phi_y, phi_z
    )
end
```

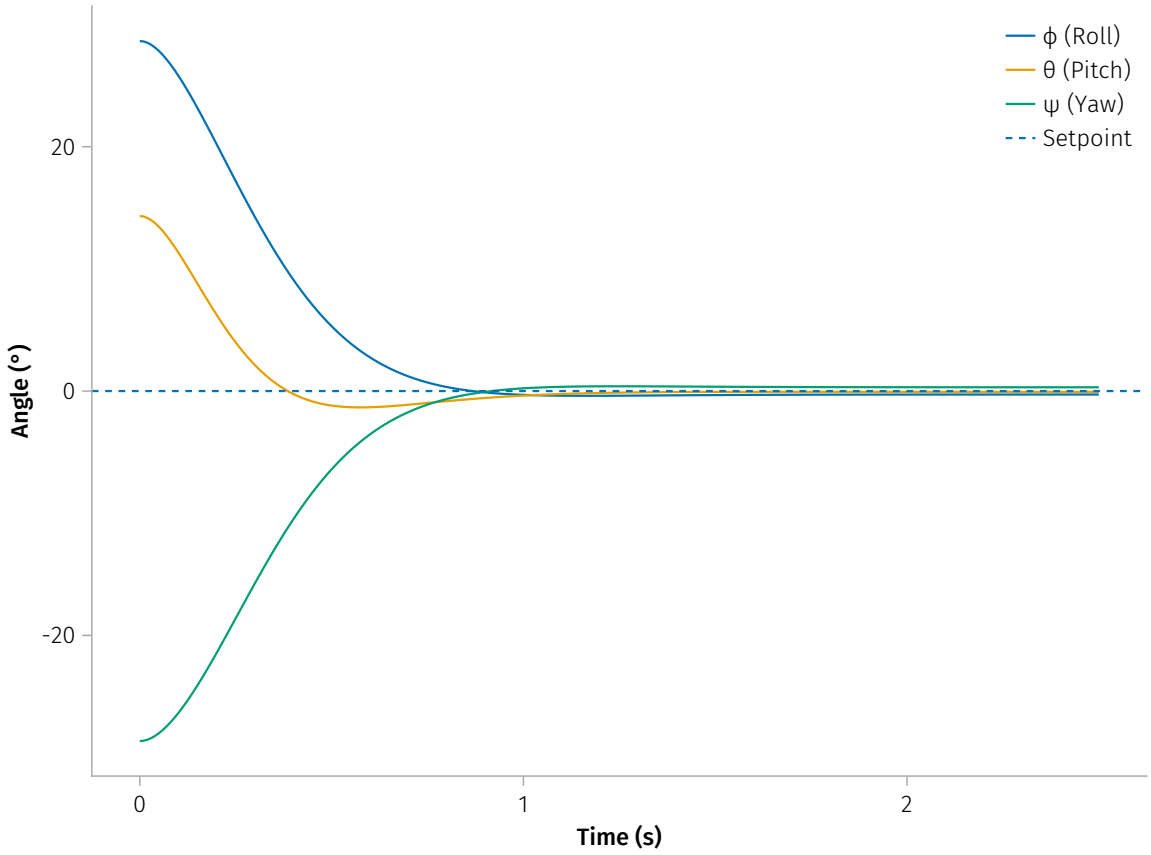
```
@named sc = SpacecraftAttitude(u0=[0.5, 0.25, -0.5])

@named setpoint_sca = B.Constant(k=0)

@named feedback_  = B.Feedback()
@named feedback_  = B.Feedback()
@named feedback_  = B.Feedback()

@named ctrl_  = B.PID(k=10.0, Td=32.0, Ti=100)
@named torque_  = Rot.Torque()

@named ctrl_  = B.PID(k=10.0, Td=32.0, Ti=100)
@named torque_  = Rot.Torque()
```

```
@named ctrl_  = B.PID(k=10.0, Td=32.0, Ti=100)
@named torque_  = Rot.Torque()

sca_eqs = [
    connect(setpoint_sca.output, feedback_ .input1),
    connect(feedback_ .output, ctrl_ .err_input),
    connect(ctrl_ .ctr_output, sc.Mx),
    connect(sc.phi_x, feedback_ .input2),

    connect(setpoint_sca.output, feedback_ .input1),
    connect(feedback_ .output, ctrl_ .err_input),
    connect(ctrl_ .ctr_output, sc.My),
    connect(sc.phi_y, feedback_ .input2),

    connect(setpoint_sca.output, feedback_ .input1),
    connect(feedback_ .output, ctrl_ .err_input),
    connect(ctrl_ .ctr_output, sc.Mz),
    connect(sc.phi_z, feedback_ .input2),
]
```

```
@named sca_model = ODESystem(sca_eqs, t; systems = [
    sc, setpoint_sca,
    feedback_ , ctrl_ , torque_ ,
    feedback_ , ctrl_ , torque_ ,
    feedback_ , ctrl_ , torque_ ,
])

sca_sys = structural_simplify(sca_model)

sca_prob = ODEProblem(sca_sys, [], (0, 2.5), [])
sca_sol = solve(sca_prob, Tsit5())
```

```
times = 0:0.01:2.5
nonlinear_interp = sca_sol(times)

fig1 = Figure()
ax1 = Axis(fig1[1,1], xlabel="Time (s)", ylabel="Angle (°)")

lines!(ax1, times, rad2deg.(nonlinear_interp[sc. ]), label="  (Roll)")
lines!(ax1, times, rad2deg.(nonlinear_interp[sc. ]), label="  (Pitch)")
```

```
lines!(ax1, times, rad2deg.(nonlinear_interp[sc.  ]), label="  (Yaw)")

hlines!(ax1, [0.0]; label="Setpoint", linestyle=:dash)

axislegend(ax1)

fig1
```



Comparison with Linear Model

```
@component function LinearSpacecraftAttitude(
    ; name, Jx=100.0, Jy=100.0, Jz=100.0, u0=[0.0,0.0,0.0],  0=[0.0,0.0,0.0], ˙0=[0.0,0.0,0
)
```

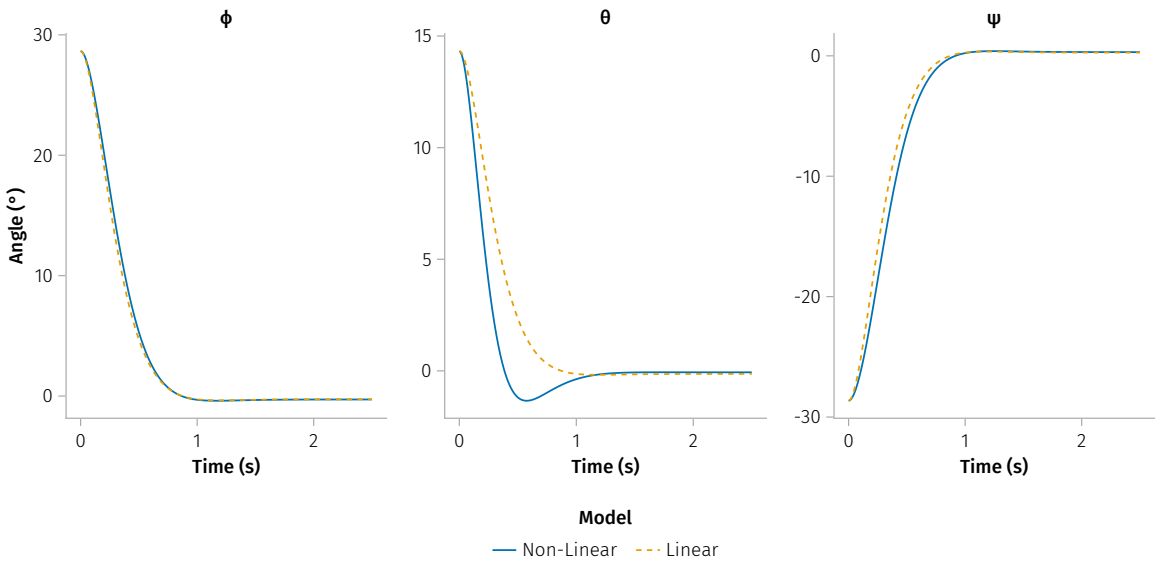
```
@named Ix = Rot.Inertia(J=Jx, phi_start=u0[1], w_start=0[1], a_start=0[1])
@named Iy = Rot.Inertia(J=Jy, phi_start=u0[2], w_start=0[2], a_start=0[2])
@named Iz = Rot.Inertia(J=Jz, phi_start=u0[3], w_start=0[3], a_start=0[3])

@named x_flange_a = Rot.Flange()
@named y_flange_a = Rot.Flange()
@named z_flange_a = Rot.Flange()

@named x_flange_b = Rot.Flange()
@named y_flange_b = Rot.Flange()
@named z_flange_b = Rot.Flange()

@named  _sensor = Rot.AngleSensor()
@named  _sensor = Rot.AngleSensor()
@named  _sensor = Rot.AngleSensor()

ps = @parameters Jx=Jx Jy=Jy Jz=Jz u0=u0  0=0 ˙0=0

D = Differential(t)

eqs = [
    connect(x_flange_a, Ix.flange_a),
    connect(y_flange_a, Iy.flange_a),
    connect(z_flange_a, Iz.flange_a),

    connect(Ix.flange_b, x_flange_b),
    connect(Iy.flange_b, y_flange_b),
    connect(Iz.flange_b, z_flange_b),

    connect(x_flange_b,  _sensor.flange),
    connect(y_flange_b,  _sensor.flange),
    connect(z_flange_b,  _sensor.flange),
]

compose(
    ODESystem(eqs, t, [], ps; name = name),
    Ix, Iy, Iz,
    x_flange_a, y_flange_a, z_flange_a,
    x_flange_b, y_flange_b, z_flange_b,
     _sensor,  _sensor,  _sensor
)
```

```
end
```

```
@named scl = LinearSpacecraftAttitude(u0=[0.5, 0.25, -0.5])

scl_eqs = [
    connect(setpoint_sca.output, feedback_ .input1),
    connect(setpoint_sca.output, feedback_ .input1),
    connect(setpoint_sca.output, feedback_ .input1),

    connect(feedback_ .output, ctrl_ .err_input),
    connect(ctrl_ .ctr_output, torque_ .tau),
    connect(torque_ .flange, scl.x_flange_a),
    connect(scl. _sensor.phi, feedback_ .input2),

    connect(feedback_ .output, ctrl_ .err_input),
    connect(ctrl_ .ctr_output, torque_ .tau),
    connect(torque_ .flange, scl.y_flange_a),
    connect(scl. _sensor.phi, feedback_ .input2),

    connect(feedback_ .output, ctrl_ .err_input),
    connect(ctrl_ .ctr_output, torque_ .tau),
    connect(torque_ .flange, scl.z_flange_a),
    connect(scl. _sensor.phi, feedback_ .input2),
]

@named scl_model = ODESystem(scl_eqs, t; systems = [
    scl, setpoint_sca,
    feedback_ , ctrl_ , torque_ ,
    feedback_ , ctrl_ , torque_ ,
    feedback_ , ctrl_ , torque_ ,
])

scl_sys = structural_simplify(scl_model)

scl_prob = ODEProblem(scl_sys, [], (0, 2.5), [])
scl_sol = solve(scl_prob, Tsit5())
```

```
fig2 = Figure(resolution=(1000,500))
ax21 = Axis(fig2[1,1], xlabel="Time (s)", ylabel="Angle (°)", title=" ")
```

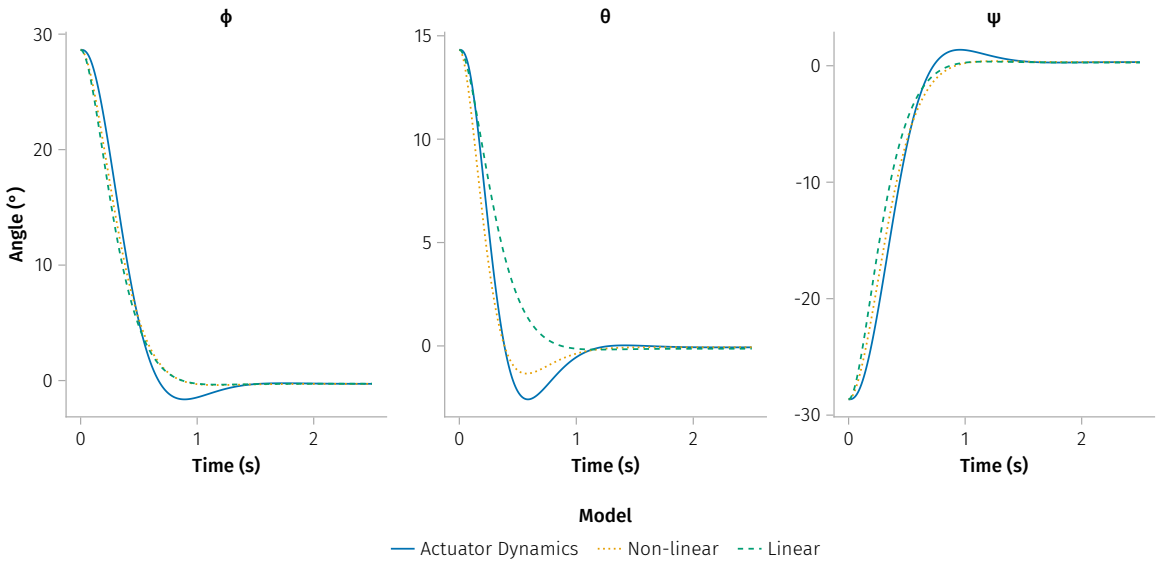
```
linear_interp = scl_sol(times)

lines!(ax21, times, rad2deg.(nonlinear_interp[sc. ]))
lines!(ax21, times, rad2deg.(linear_interp[scl.Ix.phi]), linestyle=:dash)

ax22 = Axis(fig2[1,2], xlabel="Time (s)", title=" ")

lines!(ax22, times, rad2deg.(nonlinear_interp[sc. ]))
lines!(ax22, times, rad2deg.(linear_interp[scl.Iy.phi]), linestyle=:dash)


ax23 = Axis(fig2[1,3], xlabel="Time (s)", title=" ")

lines!(ax23, times, rad2deg.(nonlinear_interp[sc. ]), label="Non-Linear")
lines!(ax23, times, rad2deg.(linear_interp[scl.Iz.phi]), linestyle=:dash, label="Linear")

fig2[2, 2] = Legend(
    fig2, ax23, "Model", framevisible=false, orientation=:horizontal, tellwidth=false
)

fig2
```



Implementing Actuator Dynamics

```
actuator_T = 0.05
@named ad_  = B.FirstOrder(T=actuator_T)
@named ad_  = B.FirstOrder(T=actuator_T)
@named ad_  = B.FirstOrder(T=actuator_T)

sc_ad_eqs = [
    connect(setpoint_sca.output, feedback_ .input1),
    connect(feedback_ .output, ctrl_ .err_input),
    connect(ctrl_ .ctr_output, ad_ .input),
    connect(ad_ .output, sc.Mx),
    connect(sc.phi_x, feedback_ .input2),

    connect(setpoint_sca.output, feedback_ .input1),
    connect(feedback_ .output, ctrl_ .err_input),
    connect(ctrl_ .ctr_output, ad_ .input),
    connect(ad_ .output, sc.My),
    connect(sc.phi_y, feedback_ .input2),

    connect(setpoint_sca.output, feedback_ .input1),
    connect(feedback_ .output, ctrl_ .err_input),
    connect(ctrl_ .ctr_output, ad_ .input),
    connect(ad_ .output, sc.Mz),
    connect(sc.phi_z, feedback_ .input2),
]

@named sc_ad_model = ODESystem(sc_ad_eqs, t; systems = [
    sc, setpoint_sca,
    feedback_ , ctrl_ , torque_ ,
    feedback_ , ctrl_ , torque_ ,
    feedback_ , ctrl_ , torque_ ,
    ad_ , ad_ , ad_
])

sc_ad_sys = structural_simplify(sc_ad_model)

sc_ad_prob = ODEProblem(sc_ad_sys, [], (0, 2.5), [])
sc_ad_sol = solve(sc_ad_prob)
```

```
fig3 = Figure(resolution=(1000,500))
ax31 = Axis(fig3[1,1], xlabel="Time (s)", ylabel="Angle (°)", title=" ")

ad_interp = sc_ad_sol(times)

lines!(ax31, times, rad2deg.(ad_interp[sc. ]))
lines!(ax31, times, rad2deg.(nonlinear_interp[sc. ]), linestyle=:dot)
lines!(ax31, times, rad2deg.(linear_interp[scl.Ix.phi]), linestyle=:dash)

ax32 = Axis(fig3[1,2], xlabel="Time (s)", title=" ")

lines!(ax32, times, rad2deg.(ad_interp[sc. ]))
lines!(ax32, times, rad2deg.(nonlinear_interp[sc. ]), linestyle=:dot)
lines!(ax32, times, rad2deg.(linear_interp[scl.Iy.phi]), linestyle=:dash)

ax33 = Axis(fig3[1,3], xlabel="Time (s)", title=" ")

lines!(ax33, times, rad2deg.(ad_interp[sc. ]), label="Actuator Dynamics")
lines!(ax33, times, rad2deg.(nonlinear_interp[sc. ]), linestyle=:dot, label="Non-linear")
lines!(ax33, times, rad2deg.(linear_interp[scl.Iz.phi]), linestyle=:dash, label="Linear")

fig3[2, 2] = Legend(
    fig3, ax33, "Model", framevisible=false, orientation=:horizontal, tellwidth=false
)

fig3
```


ϕ
θ
ψ
Angle (°)
30
20
10
0
15
10
5
0
0
-10
-20
-30
0
1
2
Time (s)
Model
Actuator Dynamics
Non-linear
Linear