```julia
include("definitions.jl")
include("utils.jl")
include("matlab_utils.jl")
using DifferentialEquations
using Plots

function AeroForcesAndMomentsBodyStateWindCoeffs(aircraft_state::AircraftState, aircraft_surfaces::AircraftControl, wind_inertial, density, aircraft_parameters::AircraftParameters)

    rho = density #stdatmo(-aircraft_state.z)
    euler_angles = EulerAngles(aircraft_state.roll, aircraft_state.pitch, aircraft_state.yaw)
    aircraft_velocity = [aircraft_state.u,aircraft_state.v,aircraft_state.w]   #In body frame
    wind_body_frame = TransformFromInertialToBody(wind_inertial, euler_angles)
    VelocityBody = aircraft_velocity - wind_body_frame   #Va vector in body frame
    wind_angles = AirRelativeVelocityVectorToWindAngles(VelocityBody)
    Va = wind_angles.Va
    Q = 0.5*rho*Va*Va
    S = aircraft_parameters.S
    b = aircraft_parameters.b
    c = aircraft_parameters.c
    CoefficientsDict = GetCoefficients(aircraft_state, aircraft_surfaces, wind_angles, aircraft_parameters)

    #Calculate force along body X
    C_X = CoefficientsDict["CX"]
    C_T = CoefficientsDict["CT"]
    T = S*Q*C_T
    X = S*Q*C_X + T

    #Calculate force along body Y
    C_Y = CoefficientsDict["CY"]
    Y = S*Q*C_Y

    #Calculate force along body Z
    C_Z = CoefficientsDict["CZ"]
    Z = S*Q*C_Z

    #Calculate moment along body X
    C_l = CoefficientsDict["Cl"]
    G = 0.0  #As mentioned by Dr.Frew on Slack
    L = b*S*Q*C_l + G

    #Calculate moment along body Y
    C_m = CoefficientsDict["Cm"]
    M = c*S*Q*C_m

    #Calculate moment along body Z
    C_n = CoefficientsDict["Cn"]
    N = b*S*Q*C_n

    aero_force = [X,Y,Z]
    aero_moment = [L,M,N]
    return aero_force,aero_moment
end

function GravityForcesBodyState(aircraft_state::AircraftState, aircraft_parameters::AircraftParameters)

    m = aircraft_parameters.m
    g = aircraft_parameters.g
    roll = aircraft_state.roll #phi
    pitch = aircraft_state.pitch #theta

    #Calculate force along body X
    X = -m*g*sin(pitch)
    #Calculate force along body Y
    Y = m*g*cos(pitch)*sin(roll)
    #Calculate force along body Z
    Z = m*g*cos(pitch)*cos(roll)

    gravity_force = [X,Y,Z]
    return gravity_force
end

function AircraftForcesAndMoments(aircraft_state::AircraftState, aircraft_surfaces::AircraftControl, wind_inertial, density, aircraft_parameters::AircraftParameters)
    aero_force,aero_moment = AeroForcesAndMomentsBodyStateWindCoeffs(aircraft_state, aircraft_surfaces, wind_inertial, density, aircraft_parameters)
    gravity_force = GravityForcesBodyState(aircraft_state, aircraft_parameters)
    total_force = aero_force+gravity_force
    total_moment = aero_moment
    return total_force, total_moment
end

function AircraftEOM(aircraft_state,aircraft_surfaces,wind_inertial,aircraft_parameters)

    rho = stdatmo(-aircraft_state.z)
    m = aircraft_parameters.m
    euler_angles = EulerAngles(aircraft_state.roll, aircraft_state.pitch, aircraft_state.yaw)
    total_force, total_moment = AircraftForcesAndMoments(aircraft_state,aircraft_surfaces,wind_inertial,rho,aircraft_parameters)

    #Position derivatives
    velocity_vector = [aircraft_state.u, aircraft_state.v, aircraft_state.w]
    position_dot = TransformFromBodyToInertial(velocity_vector, euler_angles)

    #Euler Angle derivatives
    multiplication_matrix = GetRotationalKinematicsMatrix(euler_angles)
    roll_rate_matrix = [aircraft_state.p, aircraft_state.q, aircraft_state.r]
    euler_angles_dot = multiplication_matrix*roll_rate_matrix

    #Velocity derivatives
    u = aircraft_state.u
    v = aircraft_state.v
    w = aircraft_state.w
    u_dot = (aircraft_state.r*v) - (aircraft_state.q*w) + (total_force[1]/m)
    v_dot = (aircraft_state.p*w) - (aircraft_state.r*u) + (total_force[2]/m)
    w_dot = (aircraft_state.q*u) - (aircraft_state.p*v) + (total_force[3]/m)
    velocity_dot = [u_dot,v_dot,w_dot]

    #Rate of rotation derivatives
    Gamma = GetGammaValues(aircraft_parameters)
    p_dot = (Gamma[1]*aircraft_state.p*aircraft_state.q) - (Gamma[2]*aircraft_state.q*aircraft_state.r) +
              (Gamma[3]*total_moment[1]) + (Gamma[4]*total_moment[3])
    q_dot = (Gamma[5]*aircraft_state.p*aircraft_state.r) - (Gamma[6]*(aircraft_state.p^2 - aircraft_state.r^2)) +
              (total_moment[2]/aircraft_parameters.Iy)
    r_dot = (Gamma[7]*aircraft_state.p*aircraft_state.q) - (Gamma[1]*aircraft_state.q*aircraft_state.r) +
              (Gamma[4]*total_moment[1]) + (Gamma[8]*total_moment[3])
    rotation_rate_dot = [p_dot,q_dot,r_dot]

    x_dot = vcat(position_dot,euler_angles_dot,velocity_dot,rotation_rate_dot)
    return x_dot
end
```

```julia
function aircraft_dynamics!(du,u,p,t)
    aircraft_state = AircraftState(u...)
    control_inputs = AircraftControl(p[1]...)
    wind_inertial = p[2]
    aircraft_parameters = p[3]
    x_dot = AircraftEOM(aircraft_state,control_inputs,wind_inertial,aircraft_parameters)
    for i in 1:length(u)
        du[i] = x_dot[i]
    end
end

function simulate(initial_state, time_interval, controls, wind_inertial, aircraft_parameters, save_at_value=1.0)
    extra_parameters = [controls, wind_inertial, aircraft_parameters]
    prob = ODEProblem(aircraft_dynamics!,initial_state,time_interval,extra_parameters)
    sol = DifferentialEquations.solve(prob,saveat=save_at_value)
    aircraft_states = []
    for i in 1:length(sol.u)
        push!(aircraft_states,AircraftState(sol.u[i]...))
    end
    return aircraft_states
end

function PlotSimulation(time, aircraft_state_array, control_input_array, col, save_plots=false)

    statefields = fieldnames(AircraftState)
    controlfields = fieldnames(AircraftControl)

    #Extract State values
    x_pos_values = [getfield(state,:x) for state in aircraft_state_array]
    y_pos_values = [getfield(state,:y) for state in aircraft_state_array]
    z_pos_values = [getfield(state,:z) for state in aircraft_state_array]
    roll_values = [getfield(state,:roll) for state in aircraft_state_array]
    pitch_values = [getfield(state,:pitch) for state in aircraft_state_array]
    yaw_values = [getfield(state,:yaw) for state in aircraft_state_array]
    u_values = [getfield(state,:u) for state in aircraft_state_array]
    v_values = [getfield(state,:v) for state in aircraft_state_array]
    w_values = [getfield(state,:w) for state in aircraft_state_array]
    p_values = [getfield(state,:p) for state in aircraft_state_array]
    q_values = [getfield(state,:q) for state in aircraft_state_array]
    r_values = [getfield(state,:r) for state in aircraft_state_array]

    #Extract Control values
    da_values = [getfield(control,:da) for control in control_input_array]
    de_values = [getfield(control,:de) for control in control_input_array]
    dr_values = [getfield(control,:dr) for control in control_input_array]
    dt_values = [getfield(control,:dt) for control in control_input_array]

    #Position plots
    px = plot( time, x_pos_values, xlabel = "Time (in s)", ylabel="x (in m)"  )
    py = plot( time, y_pos_values, xlabel = "Time (in s)", ylabel="y (in m)" )
    pz = plot( time, z_pos_values, xlabel = "Time (in s)", ylabel="z (in m)" )
    p_pos = plot(px, py, pz, layout=(3,1))
    if(save_plots)
        savefig("./plots/position.png")
    end

    #Orientation plots
    proll = plot( time, roll_values, xlabel = "Time (in s)", ylabel="Roll (in radians)", legends=false )
    ppitch = plot( time, pitch_values, xlabel = "Time (in s)", ylabel="Pitch (in radians)", legends=false )
    pyaw = plot( time, yaw_values, xlabel = "Time (in s)", ylabel="Yaw (in radians)", legends=false )
    p_orientation = plot(proll, ppitch, pyaw, layout=(3,1))
    if(save_plots)
        savefig("./plots/orientation.png")
    end

    #Velocity plots
    pu = plot( time, u_values, xlabel = "Time (in s)", ylabel="u (in m/s)", legends=false  )
    pv = plot( time, v_values, xlabel = "Time (in s)", ylabel="v (in m/s)", legends=false  )
    pw = plot( time, w_values, xlabel = "Time (in s)", ylabel="w (in m/s)", legends=false  )
    p_velocity = plot(pu, pv, pw, layout=(3,1))
    if(save_plots)
        savefig("./plots/velocity.png")
    end

    #Roll rate plots
    pp = plot( time, p_values, xlabel = "Time (in s)", ylabel="p (in radians/s)", legends=false  )
    pq = plot( time, q_values, xlabel = "Time (in s)", ylabel="q (in radians/s)", legends=false  )
    pr = plot( time, r_values, xlabel = "Time (in s)", ylabel="r (in radians/s)", legends=false  )
    p_rollrates = plot(pp, pq, pr, layout=(3,1))
    if(save_plots)
        savefig("./plots/angular_velocity.png")
    end

    #Control plots
    pde = plot( time, de_values, xlabel = "Time (in s)", ylabel="delta_e (in radians)", legends=false  )
    pda = plot( time, da_values, xlabel = "Time (in s)", ylabel="delta_a (in radians)", legends=false  )
    pdr = plot( time, dr_values, xlabel = "Time (in s)", ylabel="delta_r (in radians)", legends=false  )
    pdt = plot( time, dt_values, xlabel = "Time (in s)", ylabel="delta_t", legends=false )
    # pdt = plot( time, dt_values, xlabel = "Time (in s)", ylabel="delta_t", legends=false, ylim=(0.0, 0.25)  )
    p_control = plot(pde, pda, pdr, pdt, layout=(2,2))
    if(save_plots)
        savefig("./plots/controls.png")
    end

    #Plot Aircraft trajectory
    p_trajectory = plot3d([x_pos_values], [y_pos_values], [-z_pos_values], line=(:blue, 2), xlabel="x (in meters)", ylabel="y (in meters)", zlabel="z (in meters)", legend=false)
    scatter!([x_pos_values[1]],[y_pos_values[1]],[-z_pos_values[1]], color="green")
    scatter!([x_pos_values[end]],[y_pos_values[end]],[-z_pos_values[end]], color="red")
    if(save_plots)
        savefig("./plots/trajectory.png")
    end

    println("All the plots are generated and saved in the 'plots' folder.")
end

function GetStateAndControl(trim_definition::TrimDefinitionSL,trim_variables::TrimVariablesSL)

    #Parameters that don't matter
    x = y = ψ = 0.0
    #Parameters that are zero
    ϕ = v = p = q = r = 0.0
    #=
    Since there is no wind, γ = γ_a.
    Thus, pitch θ = flight path angle γ + angle of attack α
    Also, this is constant altitude flight. So, flight path angle, γ=0.
    =#
```

```julia
    θ = trim_definition.γ + trim_variables.α
    z = -trim_definition.h
    #For straight, wings-level, there is no side slip β
    β = 0.0
    wind_angles = WindAngles(trim_definition.Va,β,trim_variables.α)
    Va_vector = WindAnglesToAirRelativeVelocityVector(wind_angles)
    u = Va_vector[1]
    w = Va_vector[3]
    state = AircraftState(x,y,z,ϕ,θ,ψ,u,v,w,p,q,r)

    de = trim_variables.δe
    da = dr = 0.0
    dt = trim_variables.δt
    control = AircraftControl(de,da,dr,dt)

    return state,control
end

function GetCost(trim_definition::TrimDefinitionSL,trim_variables::TrimVariablesSL,aircraft_parameters::AircraftParameters)

    state,control = GetStateAndControl(trim_definition,trim_variables)
    wind_inertial = [0.0,0.0,0.0]
    rho = stdatmo(-state.z)
    force, moment = AircraftForcesAndMoments(state, control, wind_inertial, rho, aircraft_parameters)
    cost = norm(force,2)^2 + norm(moment,2)^2
    return cost
end

function OptimizerCostFunction(params::Vector{Float64},trim_definition::TrimDefinitionSL,aircraft_parameters::AircraftParameters)
    trim_variables = TrimVariablesSL(params...)
    cost = GetCost(trim_definition,trim_variables,aircraft_parameters)
    return cost
end

function GetTrimConditions(trim_definition::TrimDefinitionSL,aircraft_parameters::AircraftParameters)
    lower = [-pi/4,-pi/4,0.0]
    upper = [pi/4,pi/4,1.0]
    initial_tv = [0.5, 0.5, 0.5]
    results = optimize(x->OptimizerCostFunction(x,trim_definition,aircraft_parameters), lower, upper, initial_tv)
    trim_variables_list = results.minimizer
    trim_variables = TrimVariablesSL(trim_variables_list...)
    state, control = GetStateAndControl(trim_definition, trim_variables)
    return state, control, results
end

function GetStateAndControl(trim_definition::TrimDefinitionCT,trim_variables::TrimVariablesCT)

    #Parameters that don't matter
    x = y = ψ = 0.0

    #Parameters that matter
    ϕ = trim_variables.ϕ
    z = -trim_definition.h
    #=
    Since there is no wind, γ = γ_a.
    Thus, pitch θ = flight path angle γ + angle of attack α
    =#
    θ = trim_definition.γ + trim_variables.α
    α = trim_variables.α
    β = trim_variables.β
    wind_angles = WindAngles(trim_definition.Va,β,α)
    Va_vector = WindAnglesToAirRelativeVelocityVector(wind_angles)
    u = Va_vector[1]
    v = Va_vector[2]
    w = Va_vector[3]
    #=
    Slides have defined the p,q,r terms using the rate of change of coarse angle χ. It is called chi
    χ_dot = (velocity_perpendicular_to_the_cirle)/R
    velocity_perpendicular_to_the_cirle = Va*cos(γ), where γ is the flight path angle.
    =#

    # R = (trim_definition.Va^2)/(9.81*tan(ϕ))
    R = trim_definition.R
    χ_dot = ( trim_definition.Va*cos(trim_definition.γ) )/ R
    p = -sin(θ)*χ_dot
    q = sin(ϕ)*cos(θ)*χ_dot
    r = cos(ϕ)*cos(θ)*χ_dot
    state = AircraftState(x,y,z,ϕ,θ,ψ,u,v,w,p,q,r)

    de = trim_variables.δe
    da = trim_variables.δa
    dr = trim_variables.δr
    dt = trim_variables.δt
    control = AircraftControl(de,da,dr,dt)

    return state,control
end

function GetCost(trim_definition::TrimDefinitionCT,trim_variables::TrimVariablesCT,aircraft_parameters::AircraftParameters)
    state,control = GetStateAndControl(trim_definition,trim_variables)
    wind_inertial = [0.0,0.0,0.0]
    rho = stdatmo(-state.z)
    tangent_speed = trim_definition.Va*cos(trim_definition.γ)
    centripetal_acceleration = (tangent_speed*tangent_speed)/trim_definition.R
    a_desired_inertial_frame = [0.0, centripetal_acceleration, 0.0]
    euler_angles = EulerAngles(state.roll, state.pitch, state.yaw)
    a_desired_body_frame = TransformFromInertialToBody(a_desired_inertial_frame,euler_angles)
    desired_force = aircraft_parameters.m*a_desired_body_frame
    aero_force, aero_moment = AeroForcesAndMomentsBodyStateWindCoeffs(state, control, wind_inertial, rho, aircraft_parameters)
    total_force, total_moment = AircraftForcesAndMoments(state, control, wind_inertial, rho, aircraft_parameters)
    force = total_force - desired_force
    cost = norm(force,2)^2 + norm(total_moment,2)^2 + aero_force[2]^2
    return cost
end

function OptimizerCostFunction(params::Vector{Float64},trim_definition::TrimDefinitionCT,aircraft_parameters::AircraftParameters)
    trim_variables = TrimVariablesCT(params...)
    cost = GetCost(trim_definition,trim_variables,aircraft_parameters)
    return cost
end

function GetTrimConditions(trim_definition::TrimDefinitionCT,aircraft_parameters::AircraftParameters)
    lower = [-pi/4,-pi/4,0.0,-pi/4,-pi/4,-pi/4,-pi/4]
    upper = [pi/4,pi/4,1.0,pi/4,pi/4,pi/4,pi/4]
    initial_tv = [0.5,0.5,0.5,0.5,0.5,0.5,0.5]
    results = optimize(x->OptimizerCostFunction(x,trim_definition,aircraft_parameters), lower, upper, initial_tv)
    trim_variables_list = results.minimizer
```

```
    trim_variables = TrimVariablesCT(trim_variables_list...)
    state, control = GetStateAndControl(trim_definition, trim_variables)
    return state, control, results
end

filename = "ttwistor.mat"
aircraft_parameters = AircraftParameters(filename)

#Part 1 - Verifying HW2
initial_state = [100, 200, 1000, 0.05, -0.02, 2.0, 15, -2, 3, .01, .02, .03]
s = AircraftState(initial_state...)
control_input =  [-.4, .01, .02, .3]
c = AircraftControl(control_input...)
wind_inertial = [-1, -2, -3]
rho = 1.1

af, am = AeroForcesAndMomentsBodyStateWindCoeffs(s,c, wind_inertial, rho, aircraft_parameters)
println("Aero Forces : ", af)
println("Aero Moments : ", am)
tf, tm = AircraftForcesAndMoments(s,c, wind_inertial, rho, aircraft_parameters)
println("Aircraft Forces : ", af)
println("Aircraft Moments : ", am)
xd = AircraftEOM(s, c, wind_inertial,aircraft_parameters)
println("xdot : ", xd)

#Part 2 - Verifying HW3
trim_definition = TrimDefinitionSL(18.0,0.0,1655)
state, control, results = GetTrimConditions(trim_definition, aircraft_parameters)
println("SLUF: Trim Variables: ", results.minimizer)
trim_definition = TrimDefinitionCT(18.0,0.0,1655,500.0)
state, control, results = GetTrimConditions(trim_definition, aircraft_parameters)
println("Coordinated Turn: Trim Variables: ", results.minimizer)
```