```julia
include("definitions.jl")
include("utils.jl")
include("matlab_utils.jl")
include("aircraft_eom.jl")

using LinearAlgebra
using Optim

#=
****************************************************************************************************
Functions for Part 1
=#

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.
    =#
    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

#=
****************************************************************************************************
Functions for Part 2
=#

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)
case_num = 5
save_plots=true

#Part 1
if(case_num == 1)
    trim_definition = TrimDefinitionSL(18.0,0.0,1655)
    state, control, results = GetTrimConditions(trim_definition, aircraft_parameters)
    initial_state = collect(values(state))
    control_input = collect(values(control))
    wind_inertial = [0.0,0.0,0.0]
end

#Part 2
if(case_num == 2)
    trim_definition = TrimDefinitionSL(18.0,0.0,1655)
```

```
    state, control, results = GetTrimConditions(trim_definition, aircraft_parameters)
    euler_angles = EulerAngles(state.roll, state.pitch, state.yaw)
    initial_state = collect(values(state))
    wind_inertial = [10.0,10.0,0.0]
    wind_body = TransformFromInertialToBody(wind_inertial,euler_angles)
    initial_state[7] += wind_body[1]
    initial_state[8] += wind_body[2]
    initial_state[9] += wind_body[3]
    control_input = collect(values(control))
end

#Part 3
if(case_num == 3)
    trim_definition = TrimDefinitionSL(18.0,pi/18,1655)
    state, control, results = GetTrimConditions(trim_definition, aircraft_parameters)
    initial_state = collect(values(state))
    control_input = collect(values(control))
    wind_inertial = [0.0,0.0,0.0]
end

#Part 4
if(case_num == 4)
    trim_definition = TrimDefinitionCT(20.0,0.0,200,500.0)
    state, control, results = GetTrimConditions(trim_definition, aircraft_parameters)
    initial_state = collect(values(state))
    control_input = collect(values(control))
    wind_inertial = [0.0,0.0,0.0]
end

#HW 4 CT conditions
if(case_num == 5)
    trim_definition = TrimDefinitionCT(18.0,0.0,1655,500.0)
    state, control, results = GetTrimConditions(trim_definition, aircraft_parameters)
    initial_state = collect(values(state))
    control_input = collect(values(control))
    wind_inertial = [0.0,0.0,0.0]
end

time_interval = (0,500)
time_values = [i for i in 0:time_interval[2]]
trajectory_states = simulate(initial_state, time_interval, control_input, wind_inertial, aircraft_parameters)
control_array = [AircraftControl(control_input...) for i in 1:length(trajectory_states)]
PlotSimulation(time_values, trajectory_states, control_array , 'd', save_plots)
```