```
import FLOWUnsteady as uns
import FLOWVLM as vlm
import FLOWVPM as vpm

function generate_monitor(vehicle, rho, RPMref, nsteps, save_path, Vinf;
    wingmonitor_optargs=[])

    # Collect all monitors here
    monitors = []

    # -------------------- WING MONITORS ---------------------------------------
    # Reference parameters for calculating coefficients
    # NOTE: make b, ar, and qinf equals to 1.0 to obtain dimensional force
    b_ref, ar_ref = 1.0, 1.0
    qinf = 1.0
    Jref = 1.0

    # Force axis labels
    CL_lbl = "Lift (N)"
    CD_lbl = "Drag (N)"

    # Directions of force components
    L_dir = [0, 0, 1]
    D_dir = [1, 0, 0]

    # Generate function that computes wing aerodynamic forces
    calc_aerodynamicforce_fun = uns.generate_calc_aerodynamicforce(;
                    add_parasiticdrag=true,
                    add_skinfriction=true,
                    airfoilpolar="xf-n0012-il-500000-n5.csv"
                    )

    # Left wing monitor
    monitor_name = "wingL"
    leftwing_system = vlm.get_wing(vehicle.vlm_system, "wingL_system")
    leftwing_monitor = uns.generate_monitor_wing(leftwing_system, Vinf, b_ref, ar_ref,
                                rho, qinf, nsteps;
                                calc_aerodynamicforce_fun=calc_aerodynamicforce_fun,
                                save_path=save_path,
                                run_name=monitor_name,
                                figname=monitor_name,
                                CL_lbl=CL_lbl,
                                CD_lbl=CD_lbl,
                                L_dir=L_dir,
                                D_dir=D_dir,
                                wingmonitor_optargs...)

    # Right wing monitor
    monitor_name = "wingR"
```

```
    rightwing_system = vlm.get_wing(vehicle.vlm_system, "wingR_system")
    rightwing_monitor = uns.generate_monitor_wing(rightwing_system, Vinf, b_ref, ar_ref,
                                rho, qinf, nsteps;
                                calc_aerodynamicforce_fun=calc_aerodynamicforce_fun,
                                save_path=save_path,
                                run_name=monitor_name,
                                figname=monitor_name,
                                CL_lbl=CL_lbl,
                                CD_lbl=CD_lbl,
                                L_dir=L_dir,
                                D_dir=D_dir,
                                wingmonitor_optargs...)

    push!(monitors, leftwing_monitor)
    push!(monitors, rightwing_monitor)

    # ---------- OTHER MONITORS ----------

    # State-variable monitor
    statevariable_monitor = uns.generate_monitor_statevariables(; save_path=save_path)

    # ---------- CONCATENATE MONITORS ----------
    return uns.concatenate(statevariable_monitor, monitors...)

end


run_name       = "unsteady" # Name of this simulation
save_path      = "flowUS/results/" * run_name # Where to save this simulation

# Path to DegenGeom file
geom_path = "flowUS/geometry/test/test.csv"

# ---------- SIMULATION PARAMETERS ----------
# Vehicle motion
magVvehicle    = 49.7 # (m/s) vehicle velocity
frequency      = 10.0 # (Hz) oscillation frequency in flapping motion
amplitude      = 10.0  # (deg) dihedral amplitude in flapping motion

# Freestream
magVinf        = 1e-8 # (m/s) freestream velocity
rho            = 0.93 # (kg/m^3) air density
mu             = 1.85508e-5              # (kg/ms) air dynamic viscosity
speedofsound   = 342.35                  # (m/s) speed of sound
qinf           = 0.5*rho*magVvehicle^2 # (Pa) static pressure (reference)

Vinf(X, t)     = t==0 ? magVvehicle*[1,0,0] : magVinf*[1,0,0] # Freestream function
```

```
magVref        = magVvehicle # (m/s) reference velocity (for calculation purposes since freestream is zero)

# NOTE: In this simulation we will have the vehicle move while
#       the freestream is zero. However, a zero freestream can cause some
#       numerical instabilities in the solvers. To avoid instabilities, it is
#       recommended giving a full freestream velocity (or the velocity of the
#       vehicle) in the first time step `t=0`, and a negligible small velocity
#       at any other time, as shown above in the definition of `Vinf(X ,t)`.

# ----------------- WING PARAMETERS -------------------------------------------
b = 10.0                            # (m) each wing span
chord = 5.0                         # (m) each wing chord
ar = b/chord                        # aspect ratio of each wing
twist_root = 0.0                    # (deg) twist at root
twist_tip = 0.0                     # (deg) twist at tip
lambda = 0.0                        # sweep


# ----------------- SOLVER PARAMETERS -----------------------------------------

# Aerodynamic solver
VehicleType = uns.UVLMVehicle

# Time parameters
wakelength      = 2.75*b            # (m) length of wake to be resolved
ttot            = wakelength/magVref        # (s) total simulation time
nsteps          = 200               # Number of time steps

# VPM particle shedding
p_per_step      = 4                 # Sheds per time step
shed_starting   = true              # Whether to shed starting vortex
shed_unsteady   = true              # Whether to shed vorticity from unsteady loading
unsteady_shedcrit = 0.001           # Shed unsteady loading whenever circulation
                                    #  fluctuates by more than this ratio

# Regularization of embedded vorticity

lambda_vpm      = 2.0               # VPM core overlap
sigma_vlm_solver= -1                # VLM-on-VLM smoothing radius (deactivated with <0)
sigma_vlm_surf  = 0.05*b            # VLM-on-VPM smoothing radius
sigma_rotor_surf = sigma_vlm_surf   # Rotor-on-VPM smoothing radius
sigma_vpm_overwrite = lambda_vpm * magVref * (ttot/nsteps)/p_per_step # Smoothing core size
sigmafactor_vpmonvlm= 1             # Shrink particles by this factor when
                                    #  calculating VPM-on-VLM/Rotor induced velocities
vlm_rlx         = 0.5               # VLM relaxation

# VPM solver
vpm_integration = vpm.rungekutta3   # VPM temporal integration scheme
```

```
# vpm_integration = vpm.euler

vpm_viscous     = vpm.Inviscid()          # VPM viscous diffusion scheme
# vpm_viscous   = vpm.CoreSpreading(-1, -1, vpm.zeta_fmm; beta=100.0, itmax=20, tol=1e-1)

vpm_SFS         = vpm.SFS_none            # VPM LES subfilter-scale model
# vpm_SFS       = vpm.DynamicSFS(vpm.Estr_fmm, vpm.pseudo3level_positive;
#                                alpha=0.999, maxC=1.0,
#                                clippings=[vpm.clipping_backscatter])

# Wing solver: actuator surface model (ASM)
vlm_vortexsheet           = false        # Whether to spread the wing surface vorticity as a vortex sheet
(activates ASM)
vlm_vortexsheet_overlap     = 2.125        # Overlap of the particles that make the vortex sheet
vlm_vortexsheet_distribution= uns.g_pressure# Distribution of the vortex sheet
# vlm_vortexsheet_sigma_tbv = thickness*chord / 100  # Size of particles in trailing bound vortices
vlm_vortexsheet_sigma_tbv   = sigma_vpm_overwrite
# How many particles to preallocate for the vortex sheet
vlm_vortexsheet_maxstaticparticle = vlm_vortexsheet==false ? nothing : 6000000

# Wing solver: force calculation
KJforce_type              = "regular"     # KJ force evaluated at middle of bound vortices_vortexsheet
also true)
include_trailingboundvortex = false        # Include trailing bound vortices in force calculations

include_unsteadyforce       = true         # Include unsteady force
add_unsteadyforce           = false        # Whether to add the unsteady force to Ftot or to simply output it

include_parasiticdrag       = true         # Include parasitic-drag force
add_skinfriction            = true         # If false, the parasitic drag is purely parasitic, meaning no skin
friction
calc_cd_from_cl             = false        # Whether to calculate cd from cl or effective AOA
wing_polar_file             = "xf-n0012-il-500000-n5.csv"    # Airfoil polar for parasitic drag


# ----------------- 1) VEHICLE DEFINITION -------------------------------------
println("Importing geometry...")

# Import VSP Components from DegenGeom file
comp = uns.read_degengeom(geom_path);

fuselage = uns.import_vsp(comp[1])
wingL = uns.import_vsp(comp[2])
wingR = uns.import_vsp(comp[2]; flip_y=true)
verstab = uns.import_vsp(comp[4])
horstabL = uns.import_vsp(comp[5])
horstabR = uns.import_vsp(comp[5]; flip_y=true)

println("Generating vehicle...")
```

```
# Generate a separate system for each tilting wing
wingL_system = uns.vlm.WingSystem()
uns.vlm.addwing(wingL_system, "WingL", wingL)

wingR_system = uns.vlm.WingSystem()
uns.vlm.addwing(wingR_system, "WingR", wingR)

# Generate vehicle (system of all FLOWVLM objects)
system = uns.vlm.WingSystem()
uns.vlm.addwing(system, "wingL_system", wingL_system)
uns.vlm.addwing(system, "wingR_system", wingR_system)
uns.vlm.addwing(system, "HorStabL", horstabL)
uns.vlm.addwing(system, "HorStabR", horstabR)
uns.vlm.addwing(system, "VerStab", verstab)

# Generate a grid for the fuselage
fuse_grid = uns.gt.MultiGrid(3)
uns.gt.addgrid(fuse_grid, "Fuselage", fuselage)

grids = [fuse_grid,]
tilting_systems = (wingL_system, wingR_system, ) # Systems that will tilt
vlm_system = system                    # System solved through VLM solver
wake_system = system                    # System that will shed a VPM wake

vehicle = VehicleType(   system;
                         tilting_systems=tilting_systems,
                         vlm_system=vlm_system,
                         wake_system=wake_system,
                         grids=grids
                       );

# ------------ 2) MANEUVER DEFINITION ----------------------------------------

# Non-dimensional translational velocity of vehicle over time
Vvehicle(t) = [-1, 0, 0] # Vehicle is traveling in the -x direction

# Angle of the vehicle over time
anglevehicle(t) = zeros(3)

# Control inputs
angle_wingL(t) = [amplitude*sin(2*pi*frequency * t*ttot), 0, 0]  # Tilt angle of left wing
angle_wingR(t) = [-amplitude*sin(2*pi*frequency * t*ttot), 0, 0]  # Tilt angle of right wing

angle = (angle_wingL, angle_wingR, ) # Angle of each tilting system
RPM = () # RPM of each rotor system (none)

maneuver = uns.KinematicManeuver(angle, RPM, Vvehicle, anglevehicle)
```

```
# ------------- 3) SIMULATION DEFINITION ---------------------------------------

Vref = magVvehicle # Reference velocity to scale maneuver by
RPMref = 0.0 # Reference RPM to scale maneuver by
Vinit = Vref*Vvehicle(0) # Initial vehicle velocity
Winit = pi/180*(anglevehicle(1e-6) - anglevehicle(0))/(1e-6*ttot) # Initial angular velocity

# max_particles = (nsteps+1)*(vlm.get_m(vehicle.vlm_system)*(p_per_step+1) + p_per_step) #
Maximum number of particles
max_particles = 1000000

simulation = uns.Simulation(vehicle, maneuver, Vref, RPMref, ttot;
                        Vinit=Vinit, Winit=Winit)

# ------------- 4) MONITORS DEFINITIONS ----------------------------------------
# ------------- Routine for force calculation on wings
forces = []

# Calculate Kutta-Joukowski force
kuttajoukowski = uns.generate_aerodynamicforce_kuttajoukowski(KJforce_type,
                            sigma_vlm_surf, sigma_rotor_surf,
                            vlm_vortexsheet, vlm_vortexsheet_overlap,
                            vlm_vortexsheet_distribution,
                            vlm_vortexsheet_sigma_tbv;
                            vehicle=vehicle)
push!(forces, kuttajoukowski)


unsteady(args...; optargs...) = uns.calc_aerodynamicforce_unsteady(args...;
add_to_Ftot=add_unsteadyforce, optargs...)
push!(forces, unsteady)

parasiticdrag = uns.generate_aerodynamicforce_parasiticdrag(wing_polar_file;
                                            calc_cd_from_cl=calc_cd_from_cl,
                                            add_skinfriction=add_skinfriction)
push!(forces, parasiticdrag)

# Stitch all the forces into one function
function calc_aerodynamicforce_fun(vlm_system, args...; per_unit_span=false, optargs...)

    # Delete any previous force field
    fieldname = per_unit_span ? "ftot" : "Ftot"
    if fieldname in keys(vlm_system.sol)
        pop!(vlm_system.sol, fieldname)
    end

    Ftot = nothing

    for force in forces
```

```
        Ftot = force(vlm_system, args...; per_unit_span=per_unit_span, optargs...)
    end

    return Ftot
end

# Extra options for generation of wing monitors
wingmonitor_optargs = (
                    include_trailingboundvortex=include_trailingboundvortex,
                    calc_aerodynamicforce_fun=calc_aerodynamicforce_fun
                )

monitors = generate_monitor(vehicle, rho, RPMref, nsteps,
                        save_path, Vinf;
                        wingmonitor_optargs=wingmonitor_optargs)

# ------------- 5) RUN SIMULATION ---------------------------------------------
# max_particles = (nsteps+1)*(vlm.get_m(vehicle.vlm_system)*(p_per_step+1) + p_per_step) #
Maximum number of particles
max_particles = 10000000

println("Running simulation...")

uns.run_simulation(simulation, nsteps;
                # ----- SIMULATION OPTIONS -------------
                Vinf=Vinf,
                rho=rho, mu=mu,
                # ----- SOLVERS OPTIONS ----------------
                p_per_step=p_per_step,
                max_particles=max_particles,
                max_static_particles=vlm_vortexsheet_maxstaticparticle,
                vpm_integration=vpm_integration,
                vpm_viscous=vpm_viscous,
                vpm_SFS=vpm_SFS,
                sigma_vlm_surf=sigma_vlm_surf,
                sigma_rotor_surf=sigma_vlm_surf,
                sigma_vpm_overwrite=sigma_vpm_overwrite,
                sigmafactor_vpmonvlm=sigmafactor_vpmonvlm,
                vlm_vortexsheet=vlm_vortexsheet,
                vlm_vortexsheet_overlap=vlm_vortexsheet_overlap,
                vlm_vortexsheet_distribution=vlm_vortexsheet_distribution,
                vlm_vortexsheet_sigma_tbv=vlm_vortexsheet_sigma_tbv,
                vlm_rlx=vlm_rlx,
                shed_starting=shed_starting,
                shed_unsteady=shed_unsteady,
                unsteady_shedcrit=unsteady_shedcrit,
                extra_runtime_function=monitors,
                # ----- OUTPUT OPTIONS ------------------
                save_path=save_path,
```

```
run_name=run_name,)
```