```
clc;close all; clear;

h = 10;  % height of the cylinder
r_base = 1;  % base radius of the cylinder
a = 0.1;  % quadratic coefficient
num_points = 100;  % number of points for each circle
wind_direction = [1, 0, 0];  % wind direction vector (assumed to be in x-
direction)
wind_intensity = 0.1;  % wind intensity (scaling factor for bending)

% Create height values
z = linspace(0, h, 20);

% Create radii as a function of height with quadratic modulation
r = r_base + a*z.^2;

% Apply wind bending to the cylinder
X = zeros(numel(z), num_points);
Y = zeros(numel(z), num_points);
Z = zeros(numel(z), num_points);
boundary_points = cell(numel(z), 1);  % Store boundary points for each height
for i = 1:numel(z)
    theta = linspace(0, 2*pi, num_points);
    bend_factors = wind_intensity * z(i) * (cos(theta)*wind_direction(1) +
sin(theta)*wind_direction(2));  % Calculate bending factor for each point
    X(i, :) = (r(i) + bend_factors) .* cos(theta);  % Apply bending factor to
x-coordinate
    Y(i, :) = (r(i) + bend_factors) .* sin(theta);  % Apply bending factor to
y-coordinate
    Z(i, :) = z(i) * ones(1, num_points);

    % Store boundary points for each height
    boundary_points{i} = [X(i, :); Y(i, :); Z(i, :)]';
end

% Plot the bent cylinder
figure;
surf(X, Y, Z, 'FaceColor', 'blue', 'EdgeColor', 'none');
hold on;

% Plot the boundary lines
for i = 1:numel(boundary_points)
    plot3(boundary_points{i}(:, 1), boundary_points{i}(:, 2),
boundary_points{i}(:, 3), 'r');
end

xlabel('X');
ylabel('Y');
zlabel('Z');
title('Bent Cylinder with Wind Deformation and Boundary Tracking');
view(3);  % set view to 3D
hold off;
```

```
X0=[X(1,1) Y(1,1) Z(1,1) 0 0 0 0 0 0 0 0 0];
Xe=[X(15,1) Y(15+2,1) Z(15,1) 0 0 0 0 0 0 0 0 0];
syms phi(t) theta(t) psi(t)

% Transformation matrix for angular velocities from inertial frame
% to body frame
W = [ 1,  0,        -sin(theta);
    0,  cos(phi),  cos(theta)*sin(phi);
    0, -sin(phi),  cos(theta)*cos(phi) ];

% Rotation matrix R_ZYX from body frame to inertial frame
R = rotationMatrixEulerZYX(phi,theta,psi);

% Create symbolic variables for diagonal elements of inertia matrix
syms Ixx Iyy Izz

% Jacobian that relates body frame to inertial frame velocities
I = [Ixx, 0, 0; 0, Iyy, 0; 0, 0, Izz];
J = W.'*I*W;

% Coriolis matrix
dJ_dt = diff(J);
h_dot_J = [diff(phi,t), diff(theta,t), diff(psi,t)]*J;
grad_temp_h = transpose(jacobian(h_dot_J,[phi theta psi]));
C = dJ_dt - 1/2*grad_temp_h;
C = subsStateVars(C,t);

% Define fixed parameters and control inputs
% k: lift constant
% l: distance between rotor and center of mass
% m: quadrotor mass
% b: drag constant
% g: gravity
% ui: squared angular velocity of rotor i as control input
syms k l m b g u1 u2 u3 u4

% Torques in the direction of phi, theta, psi
tau_beta = [l*k*(-u2+u4); l*k*(-u1+u3); b*(-u1+u2-u3+u4)];

% Total thrust
T = k*(u1+u2+u3+u4);

% Create symbolic functions for time-dependent positions
syms x(t) y(t) z(t)

% Create state variables consisting of positions, angles,
% and their derivatives
state = [x; y; z; phi; theta; psi; diff(x,t); diff(y,t); ...
    diff(z,t); diff(phi,t); diff(theta,t); diff(psi,t)];
state = subsStateVars(state,t);

f = [ % Set time-derivative of the positions and angles
```

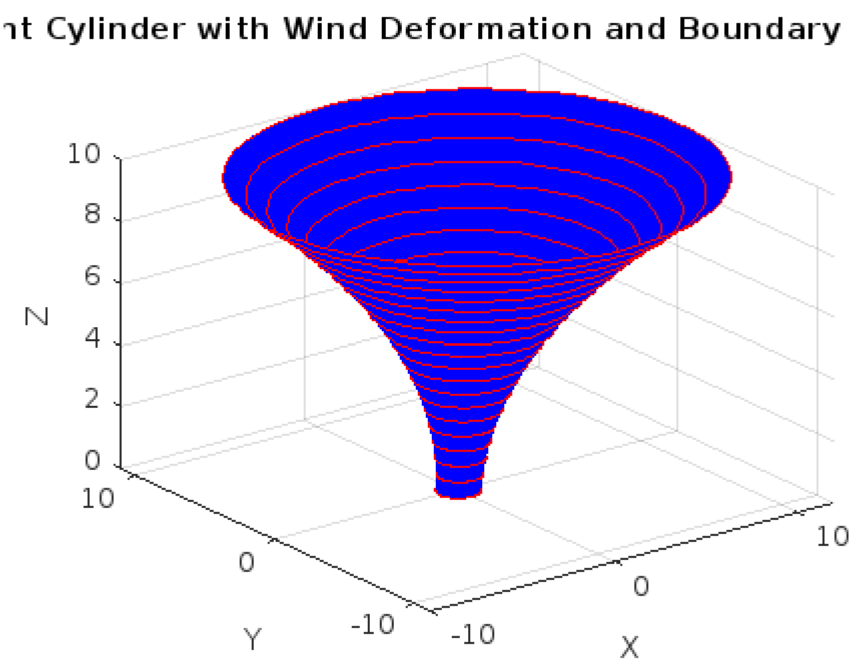
```
    state(7:12);

    % Equations for linear accelerations of the center of mass
    -g*[0;0;1] + R*[0;0;T]/m;

    % Euler–Lagrange equations for angular dynamics
    inv(J)*(tau_beta - C*state(10:12))
    ];

f = subsStateVars(f,t);

% Replace fixed parameters with given values here
IxxVal = 1.2;
IyyVal = 1.2;
IzzVal = 2.3;
kVal = 1;
lVal = 0.25;
mVal = 2;
bVal = 0.2;
gVal = 9.81;

f = subs(f, [Ixx Iyy Izz k l m b g], ...
    [IxxVal IyyVal IzzVal kVal lVal mVal bVal gVal]);
f = simplify(f);

% Calculate Jacobians for nonlinear prediction model
A = jacobian(f,state);
control = [u1; u2; u3; u4];
B = jacobian(f,control);

% Create QuadrotorStateFcn.m with current state and control
% vectors as inputs and the state time-derivative as outputs
matlabFunction(f,"File","QuadrotorStateFcn", ...
    "Vars",{state,control});

% Create QuadrotorStateJacobianFcn.m with current state and control
% vectors as inputs and the Jacobians of the state time-derivative
% as outputs
matlabFunction(A,B,"File","QuadrotorStateJacobianFcn", ...
    "Vars",{state,control});

% Confirm the functions are generated successfully
while isempty(which('QuadrotorStateJacobianFcn'))
    pause(0.1);
end


%no of state,inputs and outputs for non linear mpc controller
nx=12; %no of prediction model states
ny=12; %no of prediction model outputs
nu=4; %no of prediction model inputs

%create a non linear object whose prediction model has nx states, ny outputs,
and nu inputs, where all inputs are manipulated variables.
```

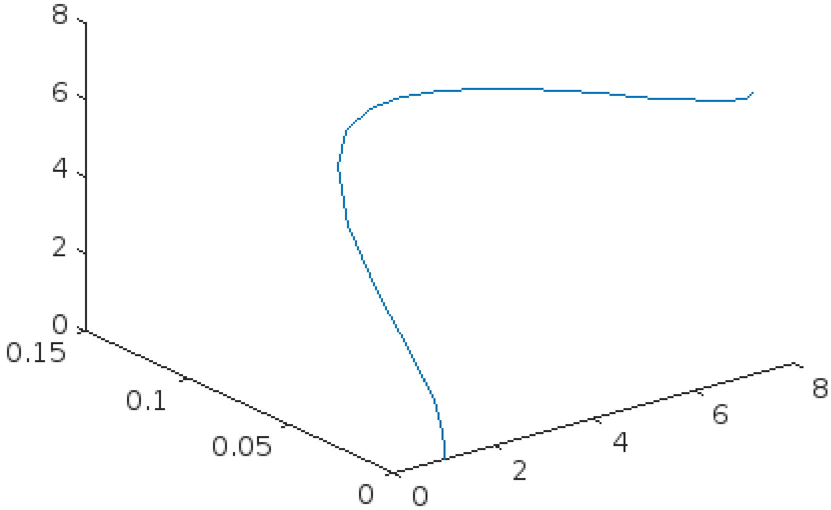
```
nlobj=nlmpc(nx,ny,nu);

%prediction model Sample time
Ts=0.3;
nlobj.Ts = Ts;

%prediction horizon steps
p=30;
nlobj.PredictionHorizon = p;

%control horizon steps
c=10;
nlobj.ControlHorizon = c;

X_end=Xe;

nlobj.Model.StateFcn = "QuadrotorStateFcn";
nlobj.Jacobian.StateFcn = @QuadrotorStateJacobianFcn;
% %Optimisation function based on minimisation of thrust inputs thereby fuel
nlobj.Optimization.CustomCostFcn = @(X,U,e,data) Ts*sum(sum(U(1:p,:)));
nlobj.Optimization.ReplaceStandardCost = true;

nlobj.Optimization.CustomEqConFcn = @(X,U,data) X(end,:)'-X_end';

% min_thrust = 0; % Minimum thrust constraint
% max_thrust = 1;

for ct = 1:nu
    nlobj.MV(ct).Min =0;% min_thrust * ones(p,1);
    nlobj.MV(ct).Max =6; %max_thrust * ones(p,1);
end

x0 = X0';
u0 = zeros(nu,1);
validateFcns(nlobj,x0,u0);

[~,~,info] = nlmpcmove(nlobj,x0,u0);

figure;
plot3(info.Xopt(:,1),info.Xopt(:,2),info.Xopt(:,3),'-')

FlyingRobotPlotPlanning(info,Ts);


function FlyingRobotPlotPlanning(Info,Ts)
Xopt = Info.Xopt;
MVopt = Info.MVopt;
fprintf('Optimal fuel consumption = %10.6f\n',Info.Cost*Ts)
t = Info.Topt;
figure();
states =
{'x','y','z','phi','theta','psi','vx','vy','vz','\dot{phi}','\dot{theta}','\do
t{psi}'};
for i = 1:size(Xopt,2)
```

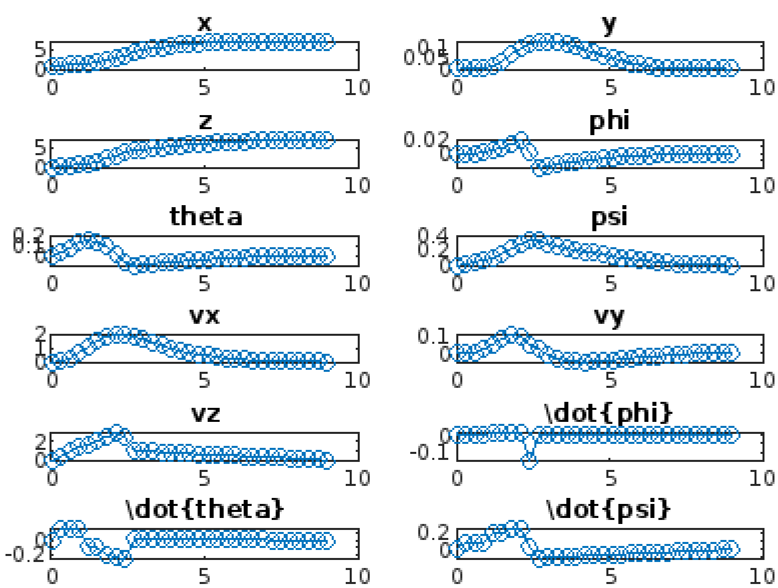
```
    subplot(6,2,i)
    plot(t,Xopt(:,i),'o-')
    title(states{i})
end
figure();
MVopt(end,:) = 0; % replace the last row u(k+p) with 0
for i = 1:4
    subplot(4,1,i)
    stairs(t,MVopt(:,i),'o-')
    axis([0 max(t) -0.1 6])
    title(sprintf('Thrust u(%i)', i));
end

end

function [Rz,Ry,Rx] = rotationMatrixEulerZYX(phi,theta,psi)
% Euler ZYX angles convention
Rx = [ 1,           0,          0;
    0,           cos(phi),  -sin(phi);
    0,           sin(phi),   cos(phi) ];
Ry = [ cos(theta),  0,          sin(theta);
    0,           1,          0;
    -sin(theta),  0,          cos(theta) ];
Rz = [cos(psi),    -sin(psi),   0;
    sin(psi),     cos(psi),   0;
    0,            0,          1 ];
if nargout == 3
    % Return rotation matrix per axes
    return;
end
% Return rotation matrix from body frame to inertial frame
Rz = Rz*Ry*Rx;
end

function stateExpr = subsStateVars(timeExpr,var)
if nargin == 1
    var = sym("t");
end
repDiff = @(ex) subsStateVarsDiff(ex,var);
stateExpr = mapSymType(timeExpr,"diff",repDiff);
repFun = @(ex) subsStateVarsFun(ex,var);
stateExpr = mapSymType(stateExpr,"symfunOf",var,repFun);
stateExpr = formula(stateExpr);
end

function newVar = subsStateVarsFun(funExpr,var)
name = symFunType(funExpr);
name = replace(name,"_Var","");
stateVar = "_" + char(var);
newVar = sym(name + stateVar);
end

function newVar = subsStateVarsDiff(diffExpr,var)
if nargin == 1
```

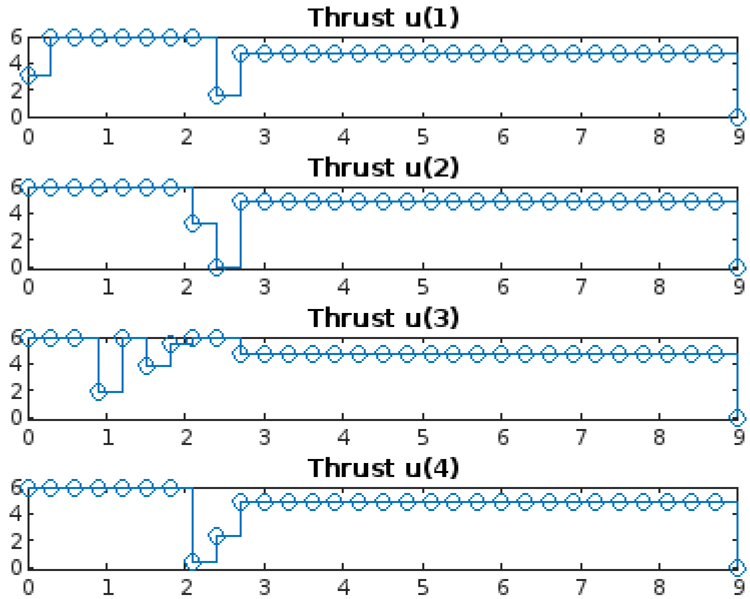
```
    var = sym("t");
end
c = children(diffExpr);
if ~isSymType(c{1},"symfunOf",var)
    % not f(t)
    newVar = diffExpr;
    return;
end
if ~any([c{2:end}] == var)
    % not derivative wrt t only
    newVar = diffExpr;
    return;
end
name = symFunType(c{1});
name = replace(name,"_Var","");
extension = "_" + join(repelem("d",numel(c)-1),"") + "ot";
stateVar = "_" + char(var);
newVar = sym(name + extension + stateVar);
end
```

```
Zero weights are applied to one or more OVs because there are fewer MVs than
OVs.
Model.StateFcn is OK.
Jacobian.StateFcn is OK.
No output function specified. Assuming "y = x" in the prediction model.
Optimization.CustomCostFcn is OK.
Optimization.CustomEqConFcn is OK.
Analysis of user-provided model, cost, and constraint functions complete.
Slack variable unused or zero-weighted in your custom cost function.
All constraints will be hard.
Optimal fuel consumption =  53.083683
```




8
6
4
2
0
0.15
0.1
0.05
0
0
2
4
6
8


x
y
z
phi
theta
psi
vx
vy
vz
\dot{phi}
\dot{theta}
\dot{psi}



Published with MATLAB® R2024a